2019-04-04

Reusable Annotations for Matching of Event Sequences to Construct Firewall Policies

Heric Flores-Huerta
Worcester Polytechnic Institute

Reusable Annotations for Matching of Event Sequences to Construct Firewall Policies

by Heric Flores Huerta

A Thesis Proposal

Submitted to the Faculty

of the

WORCESTER POLYTECHNIC INSTITUTE

In partial fulfillment of the requirements for the

Degree of Master of Science

in

Computer Science

by

December 2019

APPROVED:

Professor Craig A. Shue, Major Thesis Advisor

Professor Lane T. Harrison, Thesis Reader

Professor Craig E. Wills, Head of Department

1 Abstract

Organizations of all types use firewall systems to protect their networks from threats. Those firewalls are governed by the policies used to configure them. The PEACE (Policy Enforcement and Access Control for End-points) system is a new combination, network-plus-host based firewall that gives analysts a novel new set of data to build policy attributes for. This data are semi-structured strings that represent the hierarchy of graphical user interface components that have been interacted with around the time that host sent a network request. The multivariate, hierarchical, semi-structured nature of this data can make it a laborious or non-intuitive task to create the string matching rules that are used by the firewall policies. We present a targeted, interactive, event-sequence based [8] tool for the purpose of building policies for the PEACE firewall system's graphical user interface data.

2 Introduction

IT professionals focusing on managing risk at organizations of all sizes protect themselves from threats on their networks by using firewalls and building the policies necessary to classify packets at machine speeds. These IT professionals can be managed service providers providing services to small businesses or an infrastructure engineer managing risk within their own company's network. These network administrators have a stake in ensuring that the networks they manage are secured from unnecessary threats. For this reason they need firewalls, centralized systems designed to filter network traffic based on policies decided on at the administrative level. Basic firewalls allow network administrators to construct those policies on the network level using attributes such as source and destination IP addresses, source and destination ports, and transport layer protocols. These attributes are all simple numeric ranges or enumerated types. Network policies built with this set of attributes allows basic filtering of what kind of traffic is allowed on a network. However, there are shortcomings to such simple attributes which arise when we need to specify more complex types of network traffic, like when specifying standard known services such as FTP and SSH. A network analyst can specify they want to allow SSH traffic within their private cloud subnet by allowing TCP traffic on port 22 before running out of attributes with which to specify SSH traffic on their basic firewall. An improvement for this basic firewall would be one that does higher-level traffic analysis to glean what service is being used. This would allow the network analyst to specify port 22 TCP traffic for SSH or to disallow SSH traffic on any other port (e.g. SSH via TLS on port 443).

However, such policies can miss subtle attacks such as hijacked or spoofed applications that appear otherwise legitimate on the wire. This introduces the need for a more advanced specification of what traffic to allow on a network, to reduce the risk of allowing threats that are otherwise detectable through various kinds of traffic analysis. Performing analysis such as deep packet inspection, file analysis, behavioral analysis, domain/URL filtering, and application level analysis, newer firewalls reduce the risk imposed on their network by allowing analysts to be construct more stringent policies and thus create a tighter bound on what traffic is acceptable. This includes capabilities such as file signature filtering (e.g. virus scanning), modern application recognition (e.g. Skype, Matrix, or Bitcoin), and session based attributes. The advantage of these tighter bounds is a higher level of confidence that the traffic we allow on a network is what we intend to be allowable and thus reduces the amount of risk inherent in the network.

A new firewall system developed at WPI, called PEACE (Policy Enforcement and Access Control for End-Points), provides additional interaction data about the application's GUI (graphical user interface) context

in the form of hierarchical data flows. This data comes from the end-user's operating system and can be correlated with network flow logs. This hierarchical GUI data is stored as semi-structure strings that can take on dynamic values which can be dependent on the application, user, and test data. Policies for this new data are essentially regular expressions meant to match certain values or patterns in those GUI strings. This gives analysts more granularity (i.e. context) when creating network policies by essentially allowing them to specify the GUI components that are allowed to trigger network requests along-side the previously described restrictions on those network requests. For example, with any other firewall we can place restrictions on the Skype application's network activity to ensure it is using the correct communication protocol. With PEACE analysts can define that they want to restrict Skype's peer-to-peer protocol's network activity to the period after a user clicks the "call" button within the application. This type of granularity adds yet another layer of context to network policies that provides more security by giving analysts greater control over the behavior of individual applications. This level of context is important because periodic network requests such as updates are known and easy to build policies for, but user interactions with applications can trigger network requests. One can build policies to allow or deny all or large generalizations of an application's traffic which, if unnecessarily blocked can create disruptions in business activity, but if unnecessarily allowed, can create unavoidable security concerns. Therefore PEACE aims to create the contextual data necessary to allow network analysts to begin to create policies with greater layer of granularity not seen before.

However, due to the unstructured nature of the GUI strings consisting of potentially dynamic values, it would be difficult to construct a policy to match an entire application given a text-box a list of flows to match. Therefore, to leverage these new features, analysts could make use of tools specially built to construct and verify policies utilizing this new GUI context data.

The PEACE firewall system requires a central controller to make decisions on whether to allow or deny network flows. This controller is a server much like those of other firewall systems. Organizations also require human analysts to administer the system. The added cost of the PEACE system, over other firewall systems, is limited to the network and computational overhead. It requires sensors to be run on end-user systems for data collection and analysis (computational overhead). It also incurs a delay when that monitor requests permission to use the network from the central controller (network overhead). These have largely been seen as acceptable especially considering the benefits described above.

Our goal was to explore the work-flow of analyst firewall management tasks and how we could make improvements to the tools they use when executing those tasks. The questions involved in this exploration are: (1) what are the tasks analysts conduct in the management of firewall policies, (2) what are they trying to learn from that data, (3) what interactions do they conduct on that data do that learning, and (4) how can these interactions be improved? By asking these questions within the work-flow of policy management we steadily built a new visual system for conducting each task. Our work is centered around the exploration of the benefits of building a visual system targeted at policy management for the PEACE firewall system's GUI context data, such as how to help analysts, with no prior knowledge of this GUI data, reason and make decisions based around that data.

Visual design for human-computer interaction promises to benefit end users by making complex data-sets easier to ingest and integrate into work-flows. By allowing visual exploration of multivariate data-sets, users can interact, reason about, and make decisions on data which they nothing about. In other words, these natural visual data interactions would allow users to make policy decisions without learning yet another domain specific language for querying that data and allow them to maintain confidence that those policies gain complete coverage over the data-set without the need to sift through or query large network data-sets.

We used these concepts to create a visual system targeted for the design of GUI data policies on the PEACE firewall system. Within the system we include three views, each one targeted at a different task in the work-flow of creating network policies. Those views are: (1) the flow filter view used to construct filters to obtain targets flows, (2) the policy view to create and edit policies, and (3) the GUI-data rule builder to build rules on the GUI-data to include in the more general PEACE firewall policies.

Our work is unique in the academic and practitioner space because it combines visualizations and policy construction. By constructing an integrated visualization system for the process of building network firewall policies, we are effectively combining several disjoint areas of research into one novel application. We have targeted several research questions to narrow the focus of our work on the GUI-data rule builder: (1) How do we construct an effective visual policy building system? (2) How fast can an analyst build policies with unknown data using this policy builder (3) How effective are those policies at matching desired flows?

3 Background and Related Work

Our research questions along with the specific tasks a network analyst will conduct on the PEACE system help to guide both our work and the search for work and tools that address similar questions. The individual needs we are attempting to address have been explored by previous research in the fields of visual data mining, network security, and in industry products such as firewalls and data analytics products.

3.1 Firewall Products

Firewalls are typically centralized systems that enforce network policies by inspecting all traffic that passes through it. They use a set of network attributes to make decisions about what traffic to allow or deny within the networks they manage by either white-listing or black-listing defined sets of traffic. Traffic needed to conduct operations within the network can be allowed and traffic deemed to be of unnecessary risk to the organization can be denied. Within enterprise usage, these network policies are typically defined at organizational levels (i.e. human-readable network usage policies) and are implemented within firewalls since, typically, any traffic sent from any node on the network must pass through the firewall first.

A typical work-flow for a network analyst managing a firewall system involves one of three scenarios, (1) receiving new use-cases for allowing a certain type of traffic on the organization's internal network, (2) editing an existing policy, (3) migrating to a new firewall, and (4) building-out a complete set of policies on a new network. This research mainly concerns case (1) where an analyst's focus is adding policies to encompass a single new application. One such case would be where the organization finds the need for a new desktop application (e.g. video chat or instant messaging). Using Skype as an example, the organization in question would need to make high level decisions such as to whether or not to allow calls from outside the network or keep all messaging and calls internal and whether or not to allow sending files over messaging or restrict such activity to a communication channel with more oversight. Once this use-case for Skype has been well defined, a network analyst would need to take the request and figure out how to technically fulfill the new policy. This process involves figuring out how to place restrictions at the host and network level. From the host perspective, they might go about restricting the Skype application from reading any user or network files. From the network perspective, the analyst would need to figure out what attributes to set for their firewall's network policy. In this case, they would need to allow all of the organizations employee workstations to transmit and receive internal traffic across several ports and protocol (e.g. 443/TCP or 3478-3481/UDP) as defined by Skype's online support and documentation. Assuming this documentation is

complete, the analyst would go about white-listing internal Skype messaging and calls and the Skype traffic coming from employee workstations would now be allowed by the firewall.

Another case would be the on-boarding of a new service/appliance, such as a new email server. This case is similar to the previous; however, the communicating entities are different. In this case we want to allow communication between all employee workstations and our email server, as well as external communication between our email server and any host on the Internet assuming they are not on an IP black-list and they are not filtered out by policy through authentication such as defined in an organizations DMARC records. The network analyst would follow the same process to figure out what attributes to use for the new policy (e.g. port 465/SMTP), and then builds the policies necessary to satisfy the request and allow proper functioning of the email service.

This process of building policies is not always straight forward and can involve substantial research and troubleshooting. If there is no product support and documentation is insufficient, figuring out what ports and protocols an application will use can involve a read out of network logs over regular usage. In complex networks, other applications can interfere with each-other and create non-trivial results, causing involved troubleshooting sessions with complicated solutions even involving the cooperation of product vendors. This is especially true in cases where there are multiple security products at play. This motivates the need for a firewall solution that is more than just a list of policies to edit. An analysts needs to use network data collected and the tools to explore that data to create a picture of what is happening in their network. Their motivation is to keep their networks secure while avoiding down-time of any running service.

3.2 Firewall History

There is a long history of the development of firewalls and they do not always fit into neat categories, but there are several categories covering a range of features when it comes to firewalls. Firewalls started as simple packet filters to disallow traffic based on simple network attributes. As computers grew in power and network based threats began to be recognized [4, 36], firewalls began to become ubiquitous [15]. This would grow to the point where network security administrators would add to the list of threats the firewall themselves [17].

A firewall is either host-based or network-based. In other words, it will either restrict what a host machine does on their host machine or restrict what host machine does on the network. Host-based firewalls exists entirely as host-based applications that can control network traffic in and out of those machines. Network-based firewalls can exist, largely, as centralized controllers, whereby all traffic within the network is passed through the firewall appliance so that it may make decisions on all the packets that come out of hosts on that network. Beyond this distinction, categorization may vary, but we break them down into (1) packet filters, (2) stateful/session filters, (3) application layer filters, and (4) next-generation firewalls. With the extra distinction where we argue that the PEACE firewall deserves a category all its own.

3.2.1 Packet Filters

Packet filters, or "first generation" firewalls provide network analysts with the most basic level of administration. They typically provide source and destination IP addresses, source and destination ports, and network protocol information as attributes for building policies [26, 25]. A single policy will fill these fields with the values required to describe the traffic the analyst wishes to allow or deny. This provides basic controls against what devices (or groups of devices) can communicate over defined ports on a network, which was

fine for the time since most communication was over TCP or UDP on well defined ports.

There is, however, much room for improvement from this type of firewall. Being amongst the first firewall implementations, these earlier versions did not check the stateful information of protocols such as TCP or HTTP, or even excluded the protocol field from policies altogether. There is also little to no method of validating applications, so to professionals of today it would be trivial to see that two machines on both ends using telnet on an non-standard port would bypass a policy to disallow telnet. Also, the more attributes (or context) used for inspection of network traffic, the more specific an analyst can make their firewall policy. Using more context in a rule will typically result in a more secure policy since the amount of traffic allowed is more narrow. For these reasons, successors to this type of firewall sought to introduce more context in their policies as well as improve the semantics of those policies to better describe the traffic they affect [9].

3.2.2 Stateful Filters

Stateful filters, or "second generation" firewalls would typically provide network analysts with the same level of administration as packet filters but with more verification of the protocols defined in policies. With this stateful protocol knowledge, firewalls could begin to add features such as TCP session verification or IP spoofing protection with better security than the routers and protocol itself provided at the time of the late 1980s and early 1990s [4, 5, 9].

These firewalls still lacked the ability to understand and filter applications and application content. Also, at the time stateful firewalls came to be, it became apparent that the diversity of firewall products and their individual implementations made it harder for network administrators to easily change products or even modify their rules meaning newer firewalls needed more contextual, well defined policies [5, 9].

3.2.3 Application Filters

Application filters, or "third-generation" firewalls were the next step of protection by providing stateful filters with extra understanding of well defined applications so that they could be included in firewall policies. This means that firewalls can "understand" applications (e.g. application signatures) and detect if they are being used on non-standard ports or even inspect the content of the traffic within the context of the application being used [32, 9, 11]. Application firewalls can come with knowledge of different kinds of applications and apply filters based on the individual attributes to in-going and out-going traffic on a network [5, 10, 31]. Application filters today exist as WAFs (Web Application Firewalls), database firewalls, container firewalls, and other application specific filters. Although the categorization of firewalls up to this point has placed specific features in its own bucket, they will typically include all of these features up to a certain granularity.

3.2.4 Next-Generation Firewalls

Next-generation firewalls provide even more context over which to build policies compared to the filter types described above. In addition to the policy attributes described above for packet and stateful filters as well as application filters, these firewalls provide network administrators with modern application types, application names, and a variety of source/destination host information for building even more restrictive network policies. For this reason, we call the firewalls describe above as traditional firewalls and the following firewalls "next-generation" firewalls. A modern next-generation firewall can provide host authentication (e.g. LDAP recognition), malware scanning, DNS filtering and other features to essentially integrate separate IDS features into the firewall itself. For example, Palo Alto Network's firewalls [29] are one of the most widely

used vendors of next-generation firewalls deployed in production networks. Palo Alto networks firewalls also merge IDS/IPS technologies with their firewall through tools such as signature matching, content inspection, URL/domain filtering, DNS re-writing, file inspection and conditional packet captures to name a few. This is coupled with trouble-shooting tools such as viewing live browser session logs, policy hits, and well defined log search/filtering on all views. All this combined with an API to access all these data and features, to allow analysts to build out their own features, make next-generation firewalls, such as those produced by Palo Alto Networks, powerful filters that can integrate an entire network security stack into a single interface.

Palo Alto Networks provides firewall license purchasers with vendor generated packet signatures for application based policies. Such a policy would allow packet inspection to match the signature of a particular application to make decisions against. For example, say an analyst with a default deny network policy wishes to allow Facebook's Messenger service on their network. They can attach a Facebook Messenger signature match requirement to their network policy to gain even more restrictive protection over traditional firewalls. In this scenario, a traditional firewall would be restricted to using the Facebook Messenger IP addresses (typically a manual process) and HTTPS port 443 as the policy. With more advanced domain matching, a policy can be built to add the domain name for Facebook Messenger. However, this might inadvertently allow traffic that has fallen victim to a DNS attack, man-in-the-middle attack, or a compromised Facebook web server. This next-generation firewall can then use content inspection to to ensure the actual message matches what would be expected from Facebook Messenger, making it harder for an attacker to gain entry through an external web application. In short, the goal is to be as restrictive as possible in the creation of network policies by using as much context on that traffic as possible. This allows network administrators to restrict the set of traffic they want to allow on the network and reduce the gaps the potential attackers can use to subvert those policies.

Perhaps due to the feature-rich nature of these firewall management tools, they have become complex applications requiring a considerable learning curve as evident through their multi-week, buy-in training course available to purchasers of their products. They also lack the integration of targeted visualizations in the policy building process, opting instead to present data in a tabular manner. This is a standard method of presenting/laying-out network management tool interfaces since it is common practice for administrators to use a simple text interface, although new evidence points at changing opinions and preferences towards interactive grpahical interfaces [39]. We point to these downsides as evidence that their system carries a considerable learning-curve for carrying out any management task. We aim to reduce this complexity by integrating multiple tasks into one cohesive process that uses visual cues to promote "recognition rather than recall" [39] for administrative tasks. In other words, we want to promote simplicity by the interface to more readily lead them through a work-flow instead of having them work through it by recalling series of tasks. By creating a shared visual context between policy building tasks we can reduce the amount information the analyst needs to keep in memory to carry out a single firewall management task. We point to this task coherence as evidence of reduced cognitive load, meaning lower learning curves. Our work targets the work-flows in the PEACE firewall.

3.2.5 PEACE Firewall

Compared to the Palo Alto brand of next-generation firewall products, the PEACE (Policy Enforcement and Access Control for Endpoints) firewall system has a less feature rich environment, but its power comes in the novel addition of host-based GUI data from all all internal network user workstations. The PEACE system employs the traditional, central, in-line controller to make decisions on whether to allow or deny traffic on

the network. In addition, it implements a host-based controller to collect data from the computers of human network users. This means it integrates host and network based application data to make filtering decisions, putting this compound firewall in a category by its self.

PEACE's Microsoft Windows based host-controller collects information from the operating system about user interactions. These user interactions include mouse clicks, keyboard presses, and their interactivity with graphical user interface components (i.e. GUI context data), collected from the windowing system. This user interaction data is then correlated with network connection attempts from that machine. The resulting interaction data collected may correspond to any of the network flows collected during that same period so they are rolled together into one interaction log. In other words, a single interaction log may be included in several traditional flow logs. Inside the PEACE management interface a network analyst is presented a combination of traditional, next-generation, and PEACE specific attributes for which to build a policy from. This includes basic tables to inspect flow logs and the associated GUI context data and to build policies to match those data.

This GUI data is stored as plain-text strings describing the hierarchical structure of the visual components that the user is interacting with on the screen. For example, an interaction log created from the Microsoft Excel application would look like:

Time=1609,N=Excel,CN=XLMAIN,CT=window ->
Time=1625,N=Whats New in Excel,CN=NUIDialog,CT=window ->
Time=1639,CN=NetUIHWNDElement,CT=Pane ->
Time=1649,N=Whats New in Excel,CN=NetUINetUIDialog,CT=Custom

where 'N' stands for Name, 'CN' stands for Class Name, and 'CT' stands for Content Type. Each of these attributes are application dependent values obtained from the underlying GUI's windowing system. The values they take on are dependent on the application being interacted with and can be specific to the interaction it self, especially if it is a keyboard interaction requiring user text input. Time is a delta value measured from zero to the time that the widget being described appeared on the screen. Each arrow (->) separates widgets (another level of the GUI hierarchy). The example string above represents a single GUI hierarchy, or sequence, correlated to a single flow. Multiple sequences can be attached to the same flow and are separated by double pipes (||), represented as a bold horizontal line in figure 1. This means that there is some level of uncertainty as to which sequence actually caused the network event to trigger, so all of them are rolled into one, separated by the double pipes, and stored with the single flow. As can be seen in figure 1, these GUI data strings have a semi-structure schema. In other words, they have known keys for each corresponding value (Time, Name, Class Name, and Class Type), however each key need not be present at every level of the hierarchy, and keys may exist without values. Also, the hierarchy is of undefined height, although it is non-branching. This means we can have long sequences but each one is essentially a hierarchical one-dimensional list as demonstrated in figure 2.

To build a network policy encompassing this new GUI data, a user of the PEACE firewall system needs to go about building string matching rules. A default allow rule would simply be a string-matching wild-card "*". A rule matching a specific node or entire sequence would need to include the sequence string in the correct order along with wild-cards on the ends of the sequence string to account for extra data the system may tack on such as unpredictable time values. This extra application context is valuable for network analysts trying to use as many attributes as they are given to create more stringent firewall policies. For example, in the case of creating a policy for the Skype application on our network, we now have individual

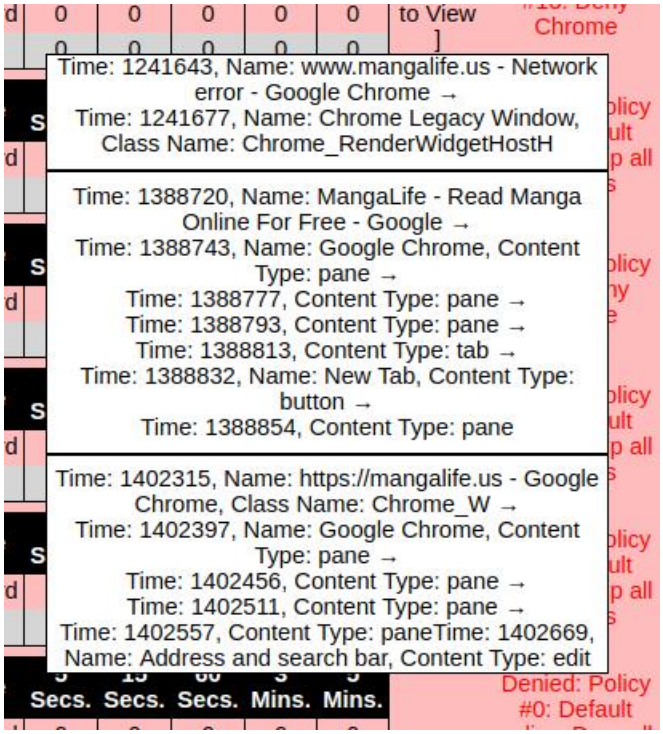


Figure 1: Old Application View Of GUI Data

graphical elements we can describe in our policy. It is now possible to restrict Skype traffic to network requests triggered by a click of the call button or a press of the enter key. Network requests for Skype services (i.e updates and server assisted peer-to-peer connections) would be easy to create policies for but we are traditionally limited in the methods for describing Skype traffic to Internet peers internal network users may want to contact. Next-generation firewalls may perform content inspection to check for malicious traffic or prevent ex-filtration of sensitive data by applications pretending to be Skype. With PEACE, an analyst can construct policies that include individual graphical components to ensure connections with external peers are purposefully created by user interactions. This makes network analysts more certain that those connections are less likely to be malicious applications masquerading as Skype and makes it harder for attackers to abuse the Skype protocol on the network.

Figure 2 shows a graphical analogy of the sequences described above. Each level of the hierarchy may contain whole, partial, or none of the attributes associated with each node in the sequence, and each attribute may hold some text or a blank value. Each node in the list shown, represents another level of the nested GUI hierarchy which begins to demonstrate the windowing system's model in a manner that is easier to ingest than the raw or even formatted GUI data strings. However, there is an unknown number of these sequences in a working set when building a single policy to match an application, so a quicker, more compact representation is needed to allow policies to be built efficiently. Figure 1 shows a snapshot of the current method that the PEACE firewall management interface uses to display GUI data. It is a dense representation constructed to be displayed next to a tabular view of similarly dense flow logs. Such a table makes it easy to skim through hundreds of flow logs and inspect individual GUI sequences, but it makes it harder to inspect similarly large amounts of GUI sequences. The need for a compact visual representation of these sequences is the motivating factor for creating a visual GUI data policy builder.

Figure 2: Visual Representation of Hierarchical GUI Data Structure

3.3 Data Analytics And Visualization Products

Production applications for the use of analytics and visualization of data from any source can be used to meet the same needs we aim to address with our targeted firewall management application. Products such as Splunk or Kibana and ElasticSearch can be set up to ingest firewall logs and maintain a copy of firewall policies. An analyst could then set up one of these applications to run statistics on flow logs and link them to those policies to begin a cross-examination of a selection of flows to policies, find anomalies in existing policies, or use their advanced filters to create new policies. Additionally, this set up could be used to create signatures intended to look for unintended traffic (i.e. look for GUI data instances in unexpected places) as extra coverage beyond what the firewall is capable of achieving in real-time.

Although it may be feasible, this work-flow requires extensive set up from expert analysts who need to create these data pipelines, signatures, policy analytics, etc. They also lack the targeted visualizations we will explore for the creation of firewall policies. It might be possible to create something usable to achieve similar goals, but we did not explore these options within our work. More importantly, this work-flow still requires a context switch from one application to another, meaning the analyst must keep flow and policy information in memory when switching between applications. We can point to previous research to show that this context switch increases cognitive load [24, 20, 28, 34] which can lead to a less effective building experience. Our aim is to tie together separate views/tasks to reduce the expected cognitive load of a context switch a help analysts more quickly arrive at the policies they wish to build.

3.4 Related Work

Previous research has explored various areas of our scope of work such as firewall policy verification, visualization of firewall policies, visualization of network and security data in general, and visual data mining. Although data visualization and visual data mining techniques for the purpose of firewall policy management is a small part of the visualization for network security field [35], they only ever come close to integrating policy creation with their data exploration tools [38] and, since PEACE is a young and novel system, none have had the opportunity to explore the new GUI data as part of their work. For this reason we also explore

more general data mining work in search of similarities to the data we use.

Early works of visualization of firewall rule-sets had the disadvantage of a smaller set of tools for visualization as well as a less defined policy semantics that would translate to those visualizations, although Firmato also attempted to address both by defining their own data model and creating visualizations off of that well defined schema [3]. Other works involving the direct visualization of firewall management data focus on visualization of entire sets of policies for the purpose of finding anomalies [38, 27, 19, 18]. The use of sunbursts in [38] allows for visualization of entire policy sets as overviews with drill-down interactions next to a more conventional tree/object view with a rule editor. The interaction between view and search proved a useful overview for gaining insights into large sets of rules but lacks the interpretation of an actual firewall (i.e. rule ordering) and might be less useful as a method for building new rules. There has also been work into exploring the visualization of other rulesets such as SELinux and SEAndroid policies [12]. This work, much like the other works just described, explores the visualization of entire rulesets and their interactions in a space that is reminiscent of the early days of firewalls where policies are mostly manually created and edited configuration files and lack the management tools to guide analysts. This is not to say the work is not useful in a more defined user graphical user environment, but its results follow the lines of the others in that they are useful overviews for sets of policies but not targeted at creating new policies for complex data sets. That being said, the policy diffing tool is something that we believe should be explored from the perspective firewall policies.

Other work in the field of data visualization for cyber security range from contextual exploration of network data for knowledge discovery, or visual pattern recognition for anomaly detection, to root-cause analysis. Work such as [14] attempt to create targeted visualization systems aimed at creating an entire work-flow for an individual task, possibly bringing together insights in a way that was not obvious with the basic representation of that network data. This shows us the value in being able to define an entire work-flow for a user with visualization-first ideals. Others, more generally, aim to provide a novel view of network data to provide context while giving the user drill down capabilities [21, 22, 40, 13, 14, 7]. It is worthwhile to understand the implications of the results presented in these works because they teach us important lessons in the design of targeted visual data mining systems. They show us that keeping the context of the data a user drills down on is important to allow them to explore large datasets without visualizations prone to becoming "hairballs" while still keeping them aware of the space they have covered or yet to cover [22, 7, 13]. Furthermore, the importance of good queries should not be understated for these same reasons (i.e. maintaining context between views) [21, 40, 7]. Overall, well defined queries and contextual views help in understanding the big picture while working with smaller data sets. In other words, it is important to keeping the user grounded and aware of their progress.

Finally, other works regarding rigorous validation of firewall policies [23, 1, 1, 16, 41, 2], such as through mathematical verifications, have also been explored but are out of the scope of our research.

3.5 Summary

Individually, these products and research address a single task we would like to explore. Security system rule management helps to guide our efforts of organizing policies in a way that makes sense to professional network analysts. Exploring previous research into visual data-mining techniques helps us to address the question of how to represent our multi-variate, semi-structured GUI data in a meaningful way. Previous firewall-specific research has guided us towards or away from areas with little research value or less useful visual implementations. Finally, previous products guide us to explore novel work in the areas of firewall

management systems and visual data analysis. It is in the intersection of these to fields that we have found novel space to begin exploring.

4 Firewall Management System

Through our work, we created an experimental web-based firewall management application that closely mimicked the functionality of the real PEACE firewall management interface. Our aim in recreating the existing system was to replicate functionality but with a user experience rooted in visual data mining techniques and to build out an integrated targeted visualization for the creation of GUI data policies. The scope of this project was to create functionality as necessary to meet the needs of the tasks that an analyst would conduct when managing the PEACE firewall system.

Management of different firewall systems may vary depending mostly on the underlying functionality and partly on the implementation of the interface to that functionality. At the most basic level a firewall management interface is giving users a method to configure firewall attributes without directly touching configuration files. Whether that is through an CLI (command line interface), API (application programming interface), or GUI (graphical user interface) can depend on the products target market and the maturity of the product or even of the individual function on that product. Up until recently, it has been assumed that most system administrators, including network administrators (i.e. network analysts) would at least use the command-line in most of their work, if not explicitly prefer it [39]. Also, network administrators may want the ability for firewalls to automate remote configuration of firewalls via its CLI or API. For these reasons, a CLI-first development approach is common amongst firewall products with the GUI catching up right after. In all of the products we have explored (e.g. Palo Alto Networks) the graphical management interfaces are made up of tabular collections of configuration values, with succinct descriptions of attribute keys, and an exhaustive amount of inputs to allow full coverage of all configuration options. This type of interface allows for a logical flow to configuring individual attributes where no functionality is, or feels, hidden from the use and is essentially an interactive visualization of the command line interface.

However, a graphical interface can induce functionality, such as selecting elements on screen to add/to or influence other elements. An example of this would be to automatically create filters when exploring tabular flows or overview visualizations. This is a tricky area to explore when regarding functionality because we do not know what features are useful to which users, large datasets can slow-down individual visualizations if they are set to automatically update after interaction triggers, and we do not want to unintentional hide a change/update to an actual configuration value just because the user clicked somewhere accidentally/unexpectedly. These side effects can frustrate users and, in the last case, have real world consequences unintended configuration changes, which is the most important side-effect to avoid when dealing with such a crucial network security device.

4.1 Analyst Tasks

As a network analyst, a user of the PEACE management interface would like to conduct several tasks typical of a firewall. It is complex to follow each task through the system since any one task can involve a process of sub-tasks tracking between each separate component of the interface. There we break down these processes into the tasks an analyst needs to do per each view in the three-view system. An analyst will be able to explore a table of flow logs and filter down to a specific subset of flows for inspection, construct a filter to match that selection of flows, query any selection of flows they have created, create a policy to target those

flows, build a GUI-policy to fit the selection and finally commit the policy to the firewall after they have verified that its effects are shown as intended.

We began with the task of creating a policy for a new application to be added to our imaginary network. Beginning with a default deny policy, the analyst's goal is to allow Skype application traffic on the network. This task would begin with finding out what basic attributes (i.e. ports, protocols, IP addresses) to fill out as part of a set of policies that exclude GUI data. Next they would need to use the application on a test machine to collect GUI data on usage of desired functions such as voice call, video chat, and instant messaging. Once all the necessary data is collected, they would build flow log filters to select only the data collected from the test run of the Skype application. Once these filters are applied, they can then sift through all the GUI data associated with those flow logs to build one or more rules to match all the sequences that appear in the data set.

This work-flow motivates the need to build out three separate views: (1) a tabular flow log view to create filters for the selection of those flows, (2) a policy view to create new policies, manage other policies that may effect the new policy, and view the effects of new policies on existing flows, and (3) a targeted GUI-policy builder view to make it easy to build policies to match GUI data that leverages the filter selections create in the flow log view. The GUI-policy builder was built separately and is the main focus of our research into visual policy building. All three views are logically interconnected by the presence of flow filters that are used by components in each view. Since an analyst begins by creating a selection of flows with which to create a policy, it follows that they would like that selection to be the working-set that follows them into the other views. This type of visual information persistence links these views logically and is part of what keeps the context of their work in the picture.

4.2 Flow View

Our flow view allows users to retrieve a typical tabular view of firewall logs. In order to use this output of logs for within the work-flow described above, the network analyst should be able to build concise filters to narrow down the amount and type of traffic they are working with when building policies around GUI text data. Any large, mid-size, and even small organization will collect enough flows to overwhelm a single visual component, be it our GUI or a CLI. In order to make this list of flows useful as a means of exploring all of the firewall logs, we would like to allow the analyst to scroll through a smaller amount of flows, page through a larger amount of flows, inspect those flows for all the data they hold, and build flow filters to create and refine their selection of flows. The analysts tasks in question consist of:

- Exploring and querying flows

  To allow users to explore flows effectively they should be able to build queries capable of returning results on numeric ranges, IP address ranges using CIDR notation, partial string matches, and negate any of those search values. Extra and interesting, functionality to explore would be turning filters on and off on a single page of results to see instant updates of filter and selecting individual rows/columns on a page to automatically build filters.

- Making selections of flows

  In order to make a selection of flows, users should be able to search for desired flows and select specific values to attach to their selection (search bar). To make this happen we explore two

solutions: 1) allow users to select any value in the table and click to add it as a filter and 2) manually construct a filter should the user know the filter values beforehand (global filters).

- Querying those selections

  Should the network analyst provide a previously known filter value then they should be able to query the selection of flows produced by that filter. For example, an analyst can use a test machine for flow collection or they may want to inspect a machine of interest as part of investigation. One of their tasks may be to inspect the flows of that particular machine. In order to do so, any query submitted within the flow view should be a query of the filtered flows. The logic of having dual querying mechanisms is to allow filters to (1) allow filters to be persistent between queries and (2) allow filters to be persistent between views. The use of these persistent filters will be shown in the following sections.

These capabilities are built into our interface an show in figure 3. Global filters have been provided as a list at the top of the page. Each filter is its own element (blue box). That the user can click to expand and be given options for filter field, filter type, and filter value. The flows produced from the search using those filters and the time range, selected by the user, provided below that, are shown in the table below both of those elements. The table itself may hold any amount of firewall logs behind it, so it will only show 100 at a time and offer the user to page into more flows. The table of results may be search using the search box, which provides an instant update of the elements in the table so the analyst may test out new filters without needing to wait for more results. The columns shown are adjustable (i.e. can be removed/added) for a more concise view of whatever the analyst may be looking for. The GUI data are hidden behind an accordion style interaction on each individual row, since the sequences are typically too long to be displayed neatly in-line, and would need a friendlier layout to be understandable anyway.

One of our goals is to speed up the creation of valid firewall policies. To achieve this goal, any interaction should offer the choice to guide a user through any process. This means providing a filter building tool targeted at the flow data provided by the firewall system thereby removing the need to learn a domain specific language for interacting with that data. Another goal of the system is to make interactions feel natural. Let us say a user decides to aggregate flows to determine the port by which to filter. They should then be able to add a port filter or a port-range filter just by clicking on table values (again reducing the learning curve). The next idea we introduced in the task of building filters is being able to run operations on filters (e.g. invert, disable, save, and annotate). These ideas are not fully explored or evaluated in our work, but are deserving of future evaluation in a real-world firewall product. Finally, persistent filters helps users keep their goals in mind with a quick glance when they advance to other views. This means they can modify their working set of flows from any view or reduce their working set to help speed up their queries in other views.

4.3 Policy View

Editing policies is the main purpose of the firewall management interface. Its entire purpose is to allow network analysts to be able to configure their allow/deny policies. In order to provide analysts with the tools necessary to build effective policies, we again introduce visual data mining techniques to provide the analyst with context as to how their changes are effecting the flows they run against. This is on top of the basic functionality needed to manage a list of firewall policies as one would from the command line.

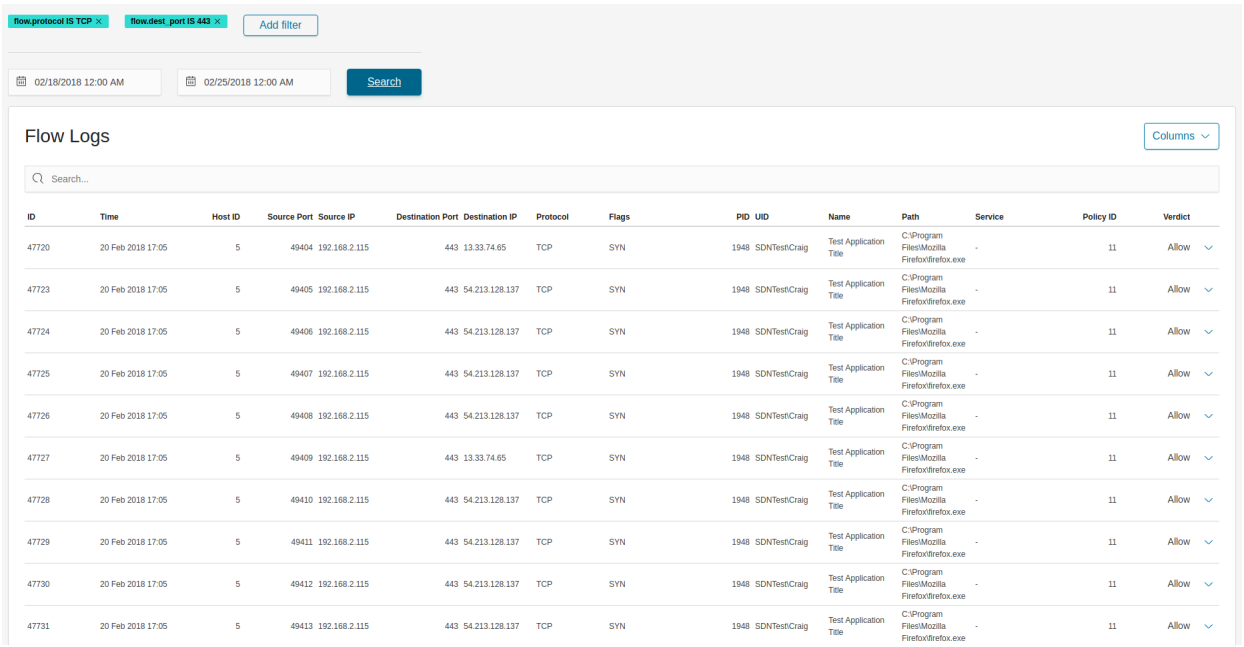


Figure 3: Graphical-User-Interface-first developed flow exploration tool. At the top, the blue bars global filters applied to the view that will also apply to other views. Below that is a global time-range filter. Below that is the main content, the table of flows with a search box for querying just the set of flows loaded on the page.

In order to build an usable management interface for firewall policy management an analysts needs to see the list of policies and the order in which they take effect, add new policies, and select a policy for editing. This is taken care of in most basic firewall interfaces. We propose, on top of this functionality, that displaying effects of edits on flows, displaying a history of those edits, and reducing the amount of information hiding will allow users to explore the policy-to-flow relationship and ensure their edits have the intended effects. The first two features would be encompassed by functionality such as a header/footer display of global edits along with short descriptions such as policy hit-count updates or a short detail view and the ability to tag those edits with a user defined description. This would allow analysts to use trial and error to learn to their edits effect the flows they have selected and allow administrators to keep a ledger of edits for review. We can reduce the amount of information being hidden by using expandable elements to show policy details and displaying interactions right on their visual components so that edits are quick and overview components can be-run against edited values without changing the view on the user or lagging the rest of the system. These three principles of design helped us make guiding choices, however, we did not explore the space of displaying a history of edits in this work, so we will only explore the area of trial and error editing and selective information hiding.

The core functionality of a firewall management interface is to construct policies to restrict the type of traffic that can be received or transmitted from any host or group of hosts on a network. Ideally, the network analyst would like these policies to be as specific as possible to reduce the risk of allowing undesirable traffic, but not too restrictive or they risk the chance of causing a disruption in availability. The correctness of these policies relies partly on the assistance of the analyst's tools (i.e. the PEACE management web application) and partly on the experience of the analyst in constructing these policies. To meet these goals, an analyst will run through several interacting sub-tasks including:

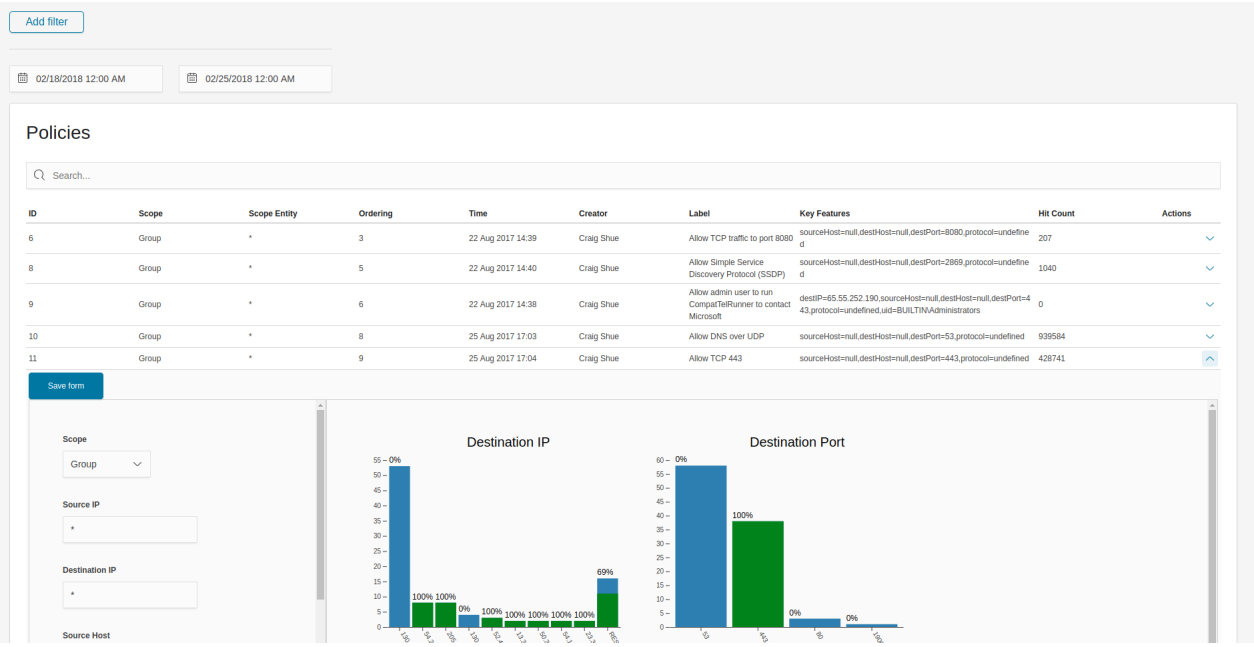


Figure 4: Graphical-User-Interface-first developed policy view and editing tool. At the top of this view are the global flow filters. Below that is the policies table. Since this is a table of values at its base, it can be queried from the search bar just like in the flow view. The drop-down or "accordion" interaction on each policy provides a view of the policy editor form. On the left of this form are the inputs for the policy's basic, single-value attributes. To the right of that panel are attribute histograms for Destination IP and Destination Port, showing the set of flows returned by the global flow filters. The coloring shows the amount of the selection of flows that are matched by the policy in its current state.

1. Search existing policies
2. Construct new policies
3. Edit existing policies

These analyst tasks are readily met by providing basic policy editing tools by providing graphical analogs of command-line tools for configuring these policies. If the analyst's task is to add a new policy to the firewall, they might begin by searching for existing policies that match the criteria their aiming for or even for a policy that is similar enough that they can edit, such as an update to an existing IP range or adding GUI matches to an existing application in the PEACE firewall. If no matches exist then they would have to go about creating a new policy. Creating a policy with the correct attribute values to match desired flows comes with experience, analysts must select attributes with regard to the traffic they wish to allow/deny while regarding other policies, the policy's order in the list, and any details of the network they are dealing with. For these reasons we add extra policy editing tools to help analysts learn the effects of their work. These additions should help analysts to (1) view the projected hit count of selected flows on all policies and (2) view how their edits effect the match distribution on their flow selection.

Recall that this task is part of the series starting with the creation of filters to construct a working set of flows to target for building our policy. So, it follows that they would want to see how their new policy, or edited policy, will effect the distribution of policy hits over all policies. If, for example, they are creating a new policy to add Skype calls on top of the pre-existing Skype update and instant-messaging policies, they would have selected the flows from an example call, created a new policy to match those flows, and check their work by ensuring the hit-count for their policy is where they expected and flows aren't hitting before or leaking down into other policies. For a network analyst to observe how any edit effects the distribution of the match on the flows they selected, we added an overview representation of that distribution that updates with every edit. This overview is seen in figure 4 as a multiple histograms showing a break-down of top values per attribute in the selected flows. Those histograms are colored in green to show the match percentage of the policy to those attributes. The two column view allows the analyst to see distribution updates as they edit their policy. Each edit to the policy or the global filters will update these histograms to reflect those changes.

4.4 GUI Policy Builder

The PEACE firewall system provides network analysts with a higher level of context, over previous generation firewalls, for use in network policies. This new data source is collected from the windowing system of the graphical user interface running on host machines in the network PEACE is running in. The host controllers running on those hosts collects data about user interactions (clicks and keyboard presses) with the graphical user interface of an application at the time a network request is created. This essentially provides network analysts with a trace of the hierarchy of graphical components the user interacted with. In other words, whenever a network connection is requested, the host controller send a connection request to the central PEACE controller along with the component the user interacted with, that component's parent, and that parent's parent, all the way down to the base application window. Since there is inherent uncertainty in which interaction caused which network request, several interactions can be correlated with a single network request. This hierarchy of graphical user interface components, or sequences, are stored as plain-text strings. Each component in that hierarchy can be thought of as an event, a component that was created and interacted

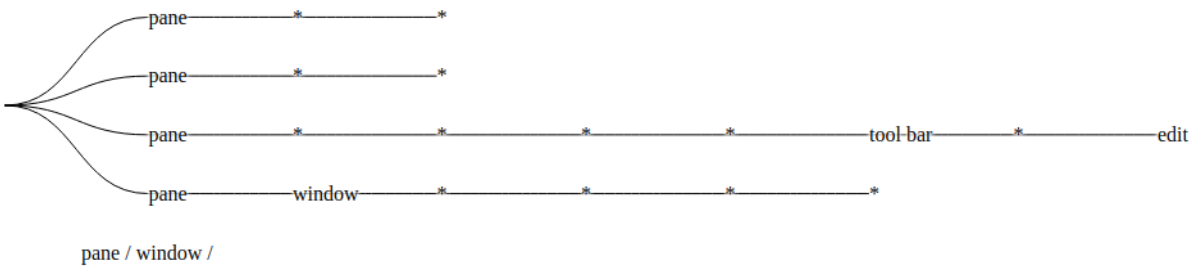


Figure 5: Early Prototype Tree of GUI Data

with at a unique time. So, the entire hierarchy can be thought of as a sequence of events, where each event is described with four attributes. An event can is described with (1) time, (2) name, (3), class name, and (4) content type attributes. For purposes of creating policies we ignore time attributes since they are always dynamic (i.e. unpredictable or non-static values). Every event is a string of these four values and every sequence is a concatenated list of these strings, with extra values to delimit different events and different sequences from each other. To build a policy allowing flows with a certain event or event-sequence, an analyst must build a wild-card capable string to match them.

These sequences of text can be used to build more descriptive and restrictive policies than traditional firewalls can. However, the multivariate and hierarchical structure of these sequences means that building a single or multiple string matching policies for an application can either be complex and non-intuitive or laborious and prone to errors if done manually. To approach this problem, we opted to build a GUI–first tool, as opposed to building command-line tools to support the task of building these GUI policies. We created an interactive visual policy building application to allow analysts to build sequence matching rules by exploring a visual representation of event-sequences. This system includes interactions to enable a network analyst to build policies by selecting sequences and individual events to automate the process of building a policy and visual cues to allow them to verify to that their work has complete coverage on their working set of flows. An early prototype of this is shown in 5. This demonstrates an attempt at showing the sequential hierarchical structure of the data, with the idea that given enough aggregate data we would be left with a single tree with an indeterminate amount of branching. We quickly narrowed down the design space towards models that were more established in the field of visual data exploration for network/security data.

We follow a couple of design guidelines when building this system including the borrowed model of event-sequence representation from Cappers and Wijk work on exploring multivariate event sequences using rules, aggregations and selections, [8] and add in our own data-specific overviews to help users gain insights into how their rules effecting the selection of sequences, again inspired from other works in the field of data-visualizatoin [37, 6]. We provide coverage summaries and use event coloring to highlight effects of user interactions and lead them towards building a proper rule-set. These event-sequences are aggregate groups of at most 1000 flows. This representation as a hierarchy or glyph-based visualization, should help users create a mental model of the data they are working with quicker than working with the strings along [20, 34]. Altogether, interactive color-updated glyphs with text-based querying and a statistical summary, we expect the user to be able to explore questions they have about the data even with-out knowledge of the underlying details of that data [30].

In our visual GUI rule building system, shown in figures 7 and 8 the event sequences are shown in a

scrollable view with different types of overview components. Each row of square glyphs represent sequences of events, with the root-parent event on the far left and the child event (highest level of the hierarchy) on the bottom. For example, the far right events would be individual components such as text-boxes and buttons while events on the far left would be base-application windows. The event sequences in these figures are aggregated and sorted by count, meaning that sequences whose attribute values are all equal are stored deduplicated and all sequences shown have unique sequences of events. Count values show how many of those duplicates are shown by that sequence.

Count values are important to analysts creating rules to match sequences because more common sequences are useful starting points since targeting those would gain the most coverage the quickest. Furthermore, very common sequences are likely a base for important interactions whereas less frequent and rare values are likely contain test-specific dynamic values or represent hierarchies of lesser used graphical components. In other words, common sequences are more likely to be foundational application components that are important for coverage. Rare events/sequences may also be network requests that were mistakenly correlated with interaction data. In any case, an analyst must be able to drill-down on each sequences and inspect the values of individual events to try to learn from the data what those hierarchies might represent. Selecting any sequence will update the detail view so users can inspect values and create rules based off those events.

A network analyst can explore the working-set of sequences by creating enough rules to cover all the flows represented by a specific application. A reasonable approach to exploring the data is to begin by gaining complete coverage of the working set to obtain a list of rules that covers all sequences and all events. Then an analyst can take this list of rules and explore what values are dynamic versus static (e.g. text-box inputs), and begin to remove data-set specific values and keep application specific values. For example, they might find "N=mozilla.orgCT=text-box,CN=input" which shows an input text-box with the user input value of "mozilla.org". In this case they would realize that the URL value is dynamic while the rest is the static representation of the URL input bar, so they would change the rule to "N=*,CT=text-box,CN=input". Once the analyst has gained an intuition of the general structure of application components, they can reduce the complete rule-set a general set of rules for representing graphical components.

Figure 6 shows the first version of the GUI builder created early on. This version uses a color scheme showing the cardinality of each individual event within the set of all sequences. In other words, all events are aggregated and counted up so that common duplicate events are shown in dark blues and the rarer events are shown in dark reds. This event-sequence view was paired with a histogram overview to show how many sequences are in the working-set. From this first prototype we learned that for most of our data, there are more duplicate sequences than duplicate events in unique sequences. This observation that most duplicates are likely entire sequences, and the need to use coloring for showing rule matching instead, lead us to decide to scrap the coloring scheme and instead aggregate by entire sequences. In the final version of our GUI builder, users can find out how common a single event is by selecting it to add it as a rule and observing the color update on all the sequences.

From that first prototype, we decided to explore the overview side of the visualization more, mostly in an attempt to increase the working set size and speed up the visualization. Figure 7 shows this effort. We added a compact overview of all the event-sequences the user was scrolling through as well as a scrubbing interaction to the histogram to allow users to filter down to a smaller working-set of sequences. Both representations of the event-sequences would update to show the new selection. Other than reducing the size of the working set, we realized that the histogram overview doesn't advance the goal of building rules, so the space could be used for something more useful and the scrubbing replace by smaller visual components such as a range

Figure 6: GUI Builder Prototype: Cardinality Coloring

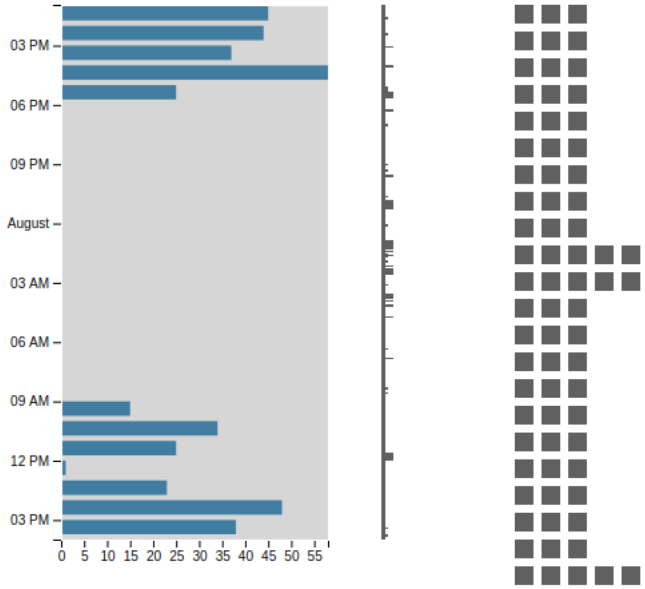


Figure 7: GUI Builder Prototype: Histogram + Scrubbing

filter.

The final prototype we explored is shown in figure 8. This version shows color highlighting for text-queries, user selected events, and histogram overviews of flow attributes. The text-box allows users to manually create rules and watch the coloring automatically update to show which events are matched by the new rule. Selecting a sequence updates the detail view to show the data in that sequence as a top-down hierarchy with the root element at the top, effectively turning the sequence 90 degrees and expanding it to show its details. Each individual event in the detail view can be selected to add the entire sequence as a rule and update the coloring in the middle view to show the interaction. The purpose of the histogram overview was to establish a shared context between the flow view, policy view and this GUI-builder view, however we observed that the context is not important for the task of building GUI rules and can probably even be complete ignored. Another lesson from this prototype is that the coloring of selected rules should match that of other rule matches to be consistent, since the fact that the rule is selected is not useful to the task of expanding coverage. Instead, the final version opts for an outline to show that a single event is selected. The final lesson we take is that we should use different colors for rules created by selecting an event versus a manual query. In the final version we even add a third color to show an event is matched by both a selected event and the current text query.

5 Development and Integration

Contexsure Network's PEACE firewall system exists as a centralized server that can be configured through a web-application management portal. As part of this research we were presented with the opportunity to build and integrate a supporting experimental visualization-focused firewall policy management tool. This tool is targeted at building rules around PEACE's GUI data source. The work presented in previous sections on the GUI builder was taken from the web-based application we built and ported to the existing PEACE

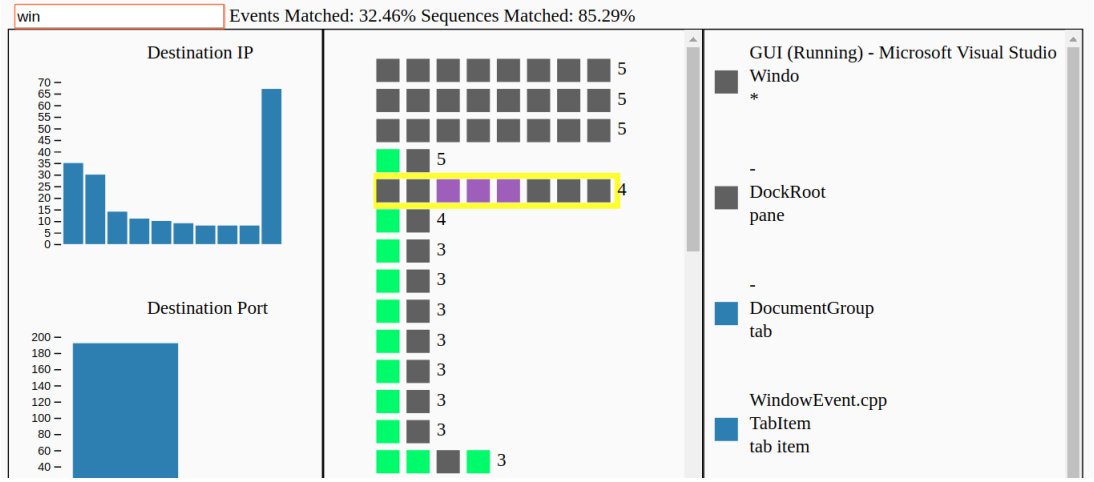


Figure 8: GUI Builder Prototype: Attribute Histograms

system for testing and eventual integration into the actual product.

5.1 Development And Integration Process

We began the exploration process by going through a series of tasks on that system to find out what improvements and additions could be implemented to meet our research goals. Once we established the three views necessary for carrying out analyst tasks on the firewall management system, we started work on our experimental web application as a web web application for viewing flow logs and policies with space for a visual GUI-policy builder constructed in-parallel but on a separate system. The web interface was built using the React Javascript library for building user interfaces, while the GUI builder was made inside of VizHub, a platform for teaching data visualization using D3.js and SVG web components. Using VizHub made it easier to create quick iteration changes and share the results through a publicly accessible web-page. The goal at this point was to re-create the views we were interested in and start making visual and functional improvements to this web management interface as a way to explore what possible interactions are needed by an analyst and what is missing from the system. Starting with our goals and a basic understand of analyst tasks, we started two separate design tasks on paper to quickly present idea on which to iterate, a process inspired by the Five Design Sheet methodology [33]. All of the design sheets we went through, and intermediate sketchs, are shown in the appendix. The first design task was to sketch a mock-up of the views we wanted in our web application and the potential features and interactions we wanted to experiment with. Our second task was to sketch out as many possible exploratory visualizations for the creation of a GUI policy builder. Figure 9 shows the page of sketches from this process. It demonstrates the breadth of different ideas we explored when trying to decide which areas were feasible and how to scope the project from the beginning. I then presented these two sets of design to a data visualization advisor and the PEACE system designer, a network administrator with experience building PEACE network policies for customers. The former will be referred to as an data visualization advisor and the latter as the client for our work. This led us into a two pronged design and development process, the first being the engineering and design of the front-end firewall management system and the second being the design and prototyping of the GUI policy builder. After presenting larger matrix of ideas to be ranked in order of feasibility and utility, we had a direction for which to begin exploring. In order to get the client and advisor all involved and contributing

to the design process we were adopted an iterative design process.

Our iterative design schedule consisted of the process of presenting new engineering and design work during weekly meetings. Every week we would discuss what potential additions and changes to focus on for development and what exploratory visualizations to think about. This meant that every week had a presentation of updated engineering work and a new set of hand-drawn designs for all views. Since the engineering was always one week behind the design process, the advisors could comment and request additions or changes to designs and then see the implementations of those designs the following week. With each iteration we targeted specific widgets (visual components) to be tweaked or remodeled, to improve functionality, improve quality of the information presented, or improve the scalability of the visualization system. The client would have a chance to comment on functionality and the advisor could guide our though process/decisions and future design ideas. On a less frequent iterative schedule, we provided functional prototypes to demonstrate interactions, run-time speeds, and implementation complexity of the design ideas we were aiming to create. After a couple of iterations, we presented the best combination of widgets and layout to begin a larger scale prototype. This process overall allowed us to explore fringe data visualizations and interactions, that may not have been feasible within the scope of a master's thesis along, and without committing to a single design choice early on.

In the end, weekly iterations were mainly composed of new design sketches of possible next-steps to explore, web programming work to meet the agreed-upon design, and a meeting with the data visualization advisor and the client to present updates and collaborate for the next iteration. This process continued until we were satisfied with the policy building experience in the experimental system and we were ready to begin porting and testing our designs on the existing PEACE system.

5.2 Existing Views

Considering the work-flow and views we targeted for our work, the existing views in the PEACE firewall management web interface includes the Recent Connections view, the List Policies view, and the Edit Existing Policy view. Figure 10 shows the current state of the recent connections view which is the equivalent of our described flow view. This top of this page shows display controls or a short list of filters available to the user. Those filters are: (1) edit the number of flows shown on the page, (2) change the origin/host of the flows fetched, (3) fetch the same flows with new filters, (4) fetch only new flows from when the page was loaded. These filters support the basic functionality for paging through flows and changing which host's traffic is returned. There are two places we targeted for improvement in this view, the first being the filters and the second being the way the GUI data is displayed. The latter improvements come in the form a GUI builder since this view, in the current version of the PEACE system, essentially serves that purpose.

Figures 11 and 12 show the current states of the policy list view and the policy editor view respectively. These are the equivalent of the policy view in our work. It is separated into two separate work-flows, the first shows a table of the policies that were already created on the firewall. It shows a quick overview of the policy, including basic policy attributes and a short description of the key features that make the allow/deny decision for that policy. In order to get a full view and edit any single policy it must be selected, bringing the user to the policy editor view. There the user gets a series of tables for editing basic policy attributes as well as interaction attributes. One of the entries on this page is the "required specific UI activity" input box for network analysts to write in their GUI rule.

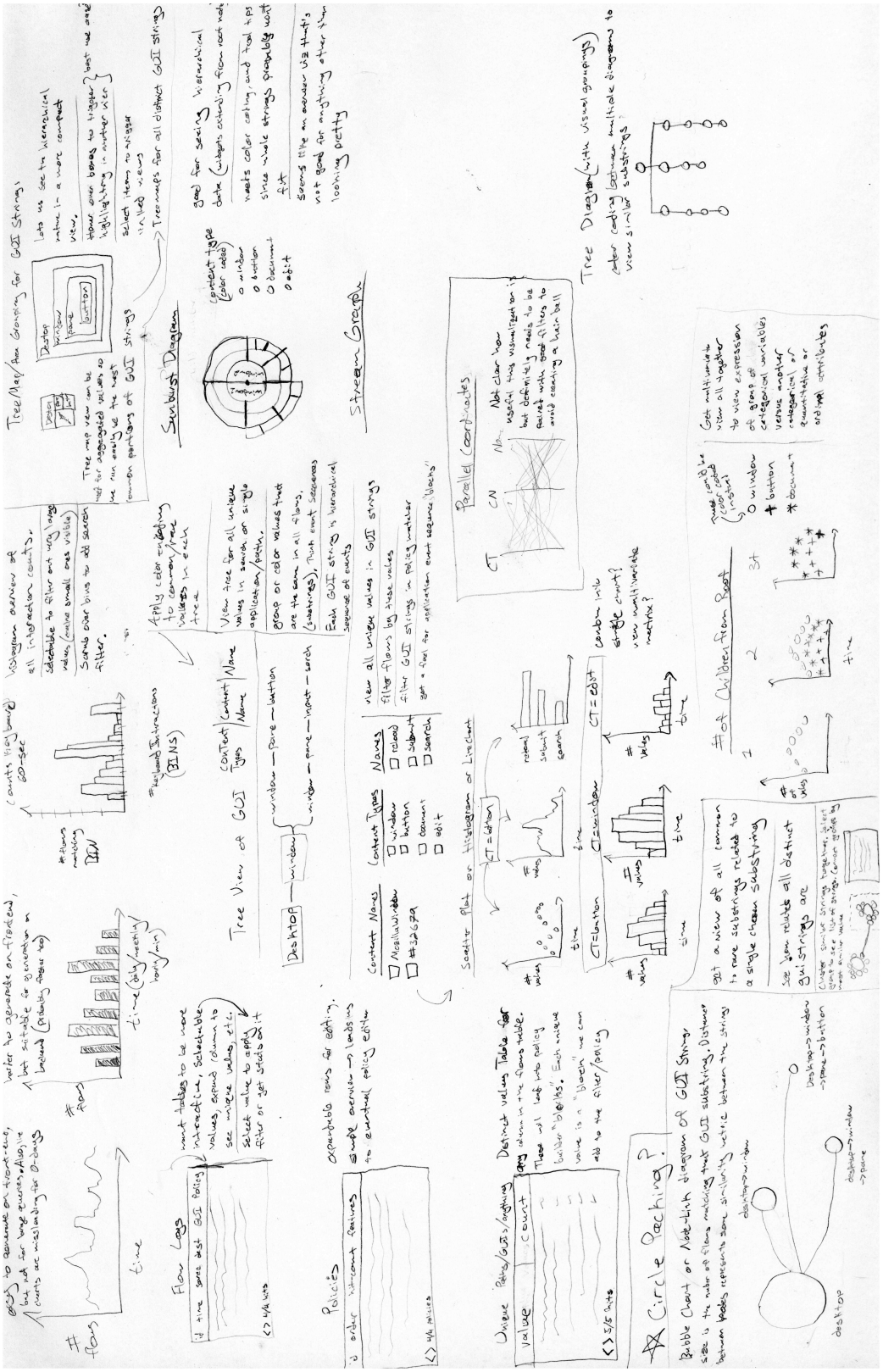


Figure 9: Exploratory Design Sketches

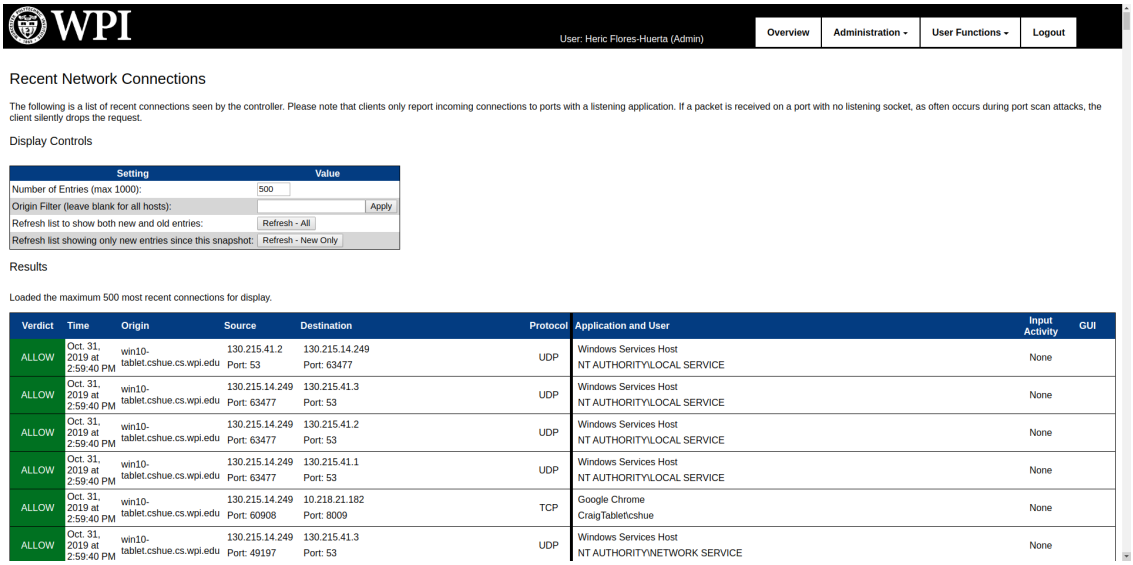


Figure 10: PEACE Interface View For Recent Connections

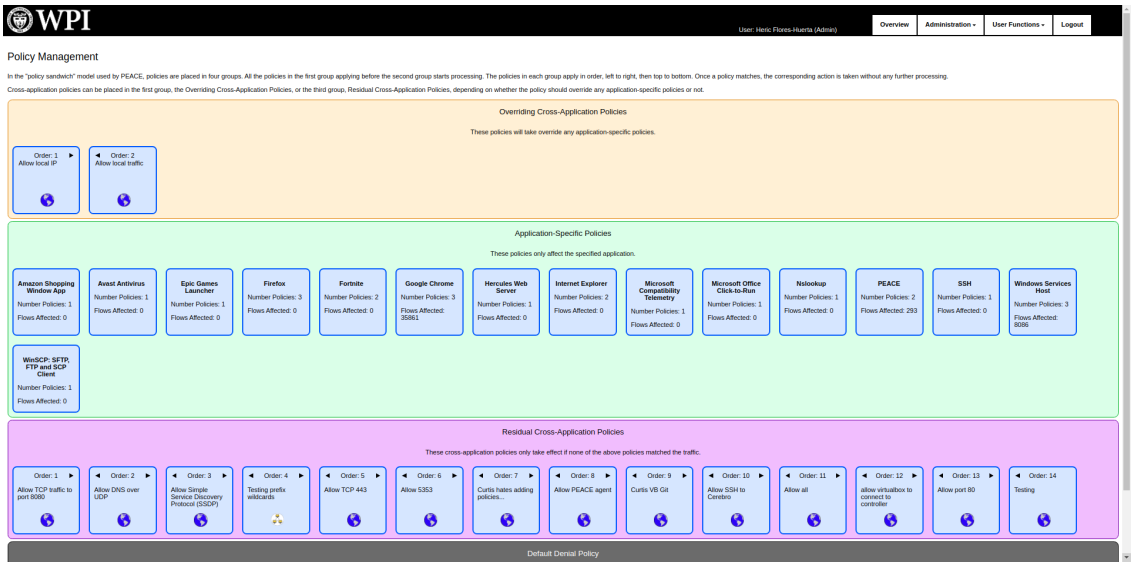


Figure 11: PEACE Interface View For Policies

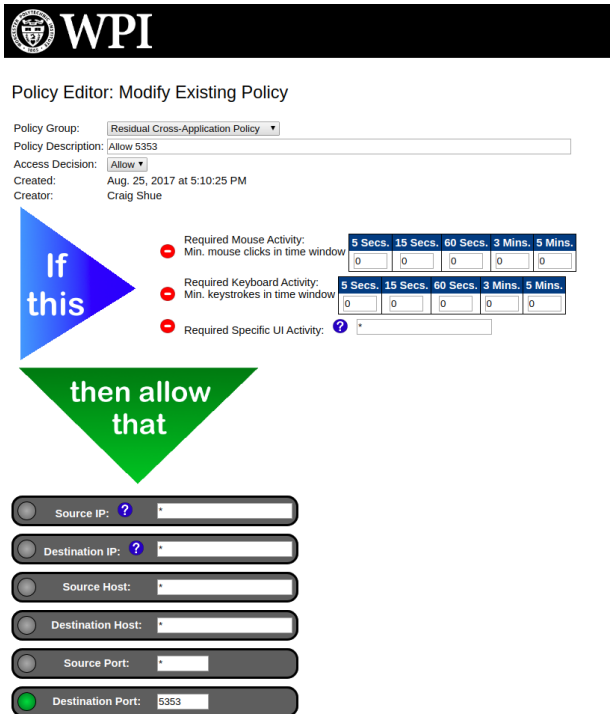


Figure 12: PEACE Interface View For Editing a Policy

5.3 Integrating New and Existing Views

As part of our work, we looked into integrating our development and design ideas into the existing PEACE system for as far as any feature would contribute to a single task the analyst would carry out on the system. Our contributions to the existing PEACE system ended up being in improvements to the GUI policy building experience. Figure 13 shows the changes we made to the flow view of the peace system. When running through the GUI policy building experiments, we realized the task was laborious and tedious and with a small amount of automation. To improve on the recent connection view, we decided to make progress towards making this a better tool for querying flows and retrieving GUI data. To accomplish this, we add filters as need while actually building GUI policies ourselves. The new list of filters includes a time filter to go back to older flows (which previously wasn't possible), and application filter to fetch only those flows associated with the application we wanted to build a policy for, and an empty-GUI filter to remove flows that didn't have any interaction data associated with them. These filters got us on the path to actually having a reasonable work-flow for building GUI policies using this interface. The final addition was the "Copy GUI Data" button which takes all the GUI sequences in the working-set of flows and copies them to the clip-board so they could be pasted to a text-editor where the rest of the rule building work was done.

The second attempt we made at improving the GUI policy building experience was the addition of an actual GUI policy builder. PEACE's new GUI policy builder, shown in figure 14, is an implementation of the event-sequence explorer we created built specifically for the PEACE firewall. This GUI builder copies all of the same filters we added to the flow view and adds them to the GUI builder, creating a sense of the shared context described in our GUI firewall management system. This version uses all of the same components and interactions. Creating new rules is done either by manually typing it into the query box and click "Add Rule", or selecting a sequence for inspection and selecting a single event in the detail view to add its

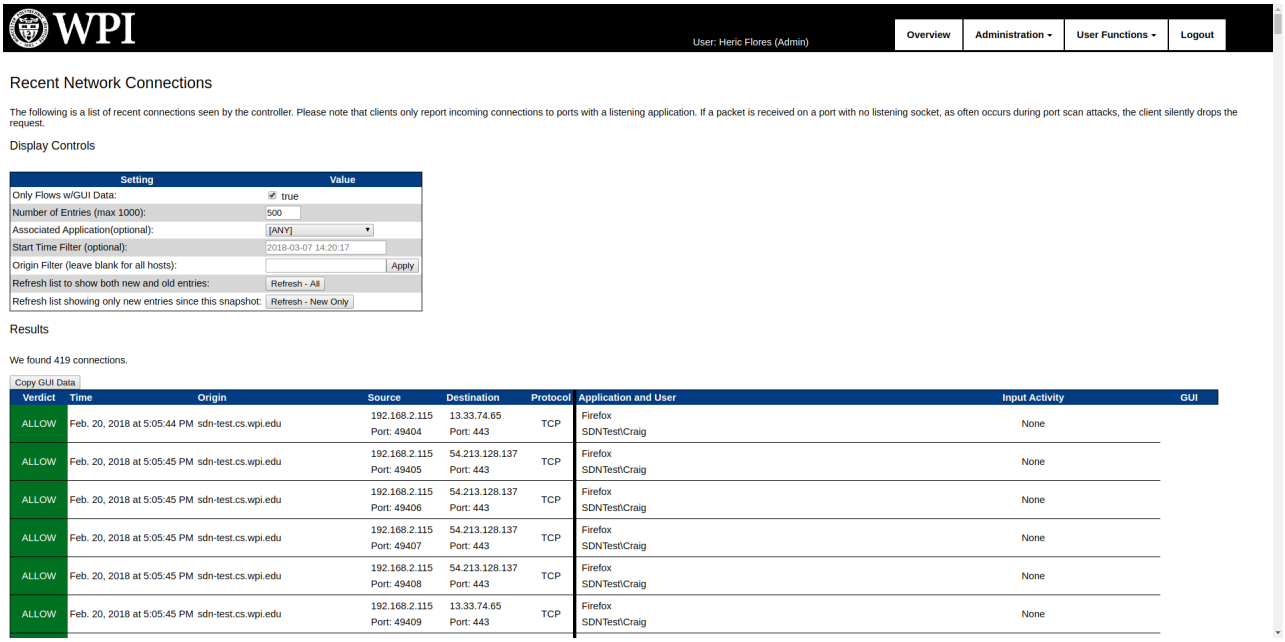


Figure 13: New Version of Recent Connections View

attributes as a rule. Adding an event in either way, or typing anything into the rule input text-box will trigger a color update on all of the event-sequences. Events matching any added rules are colored in green, rules matching the text-box content are colored in red, and rules that match both are colored in yellow. This allows users to see the effects of their rule creations in real time. There are also statistics showing the percentage of events matched and the percentage of sequences matched. Overall, this standalone tool is provides a basis for building GUI-data based policies, that would not have otherwise been possible with a network analysts' understanding of the underlying data and its structure.

5.4 Summary

Following from the previous discussion, on our observation of how visualization tools need to be targeted as improvements on everyday tasks, we sought to improve upon the PEACE firewall management interface as it existed by making incremental improvements and improving existing work-flows. Limited only by the knowledge of how a typical client of the PEACE firewall system uses the interface in their day-to-day operations, we targeted the GUI building experience. The end result was a addition of a single work-flow which was not possible beforehand.

6 Evaluation

A full evaluation of our visual system for creation of GUI-context data matching rules within network policies should include a user study in which usage data is collected as network analysts new to the PEACE firewall system are introduced to the old and new policy building work-flows. This type of user-study will be left as future work for those studying the effects of the new system and how to improve it. In this work, we ran a pilot study to collect qualitative and quantitative data by running the separate work-flows through a handful of scenarios. The goal is to collect quantitative data about system usage as well as to gain experience with

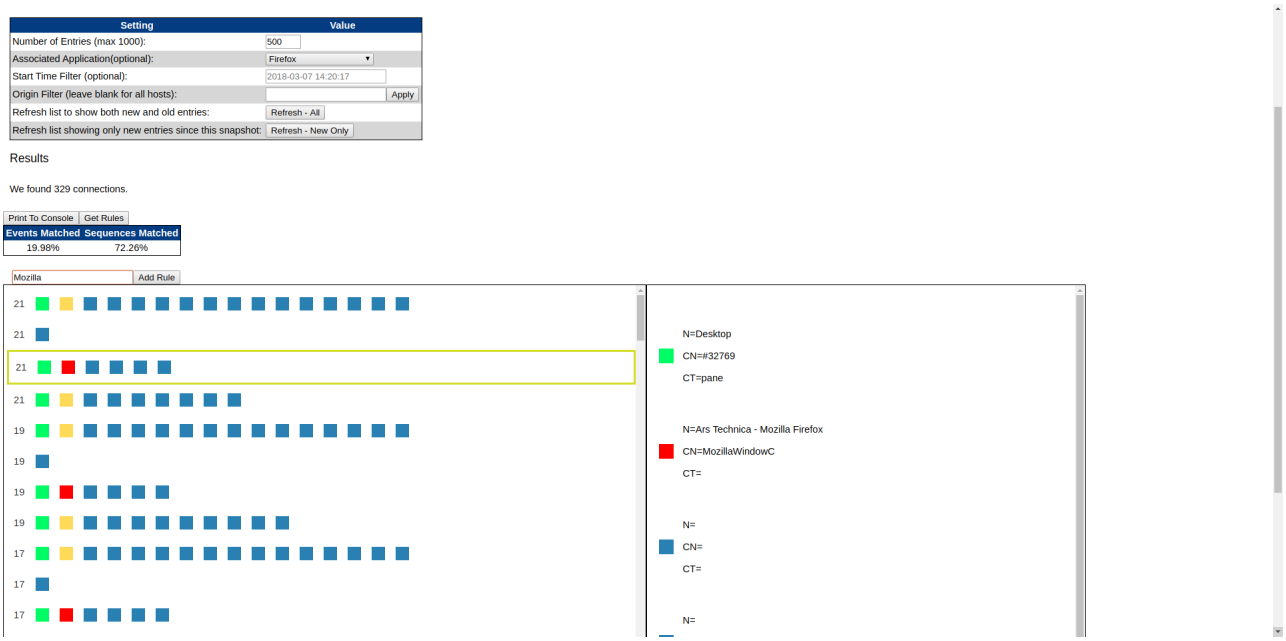


Figure 14: PEACE's New GUI Builder View

both methods to point out the key differences and their effects on how the firewall system is used.

6.1 Experimental Setup

To be fair to the old method of manual rule creation we add some functionality to the PEACE firewall system. Those changes consisted of extra flow filters, one-click copy of GUI text data for all flows displayed, and a non-hidden tabular view of all the GUI text data. The extra flow filters gave users the ability to quickly narrow down flow selections to only those of a selected application with non-empty GUI data, making it faster to retrieve all the data needed for creation of a new policy. Adding the non-hidden tabular view of all GUI text data and a button to copy it all to the clipboard made it easier to scroll through flows and copy data over to a text editor for manual rule creation.

There are two experiments in two parts used to evaluate the old and new system of building GUI data rules. The first experiment run is to create a GUI data rules for coverage of the Google Chrome application's flows. This was split into two parts, one part for testing the new visual system for building GUI data policies and another part for testing the old system. The same was done when running the study on the Firefox application. For each test-run of the rule building work-flow the data collected included clicks, elapsed time, average rule length, and the amount of overlap between rules. Overlap between rules is measured as the number of GUI data rules that apply to each event in all the event sequences. This is summed up over all sequence events in each case study and presented in table 2.

6.2 Rule Creation Statistics

This pilot study started with the smaller test of 29 network flows worth of Google Chrome data to gauge the utility of the work-flow in each case. The results of these first two tests are seen in the first two rows of table 1. In both cases complete coverage of those non-empty event-sequences, or 97% of all sequences, was

Stat	Time	Clicks	#of Rules	Events Matched	Sequences Matched
Chrome Manual	28m 30s	380	14	77.46%	97.56%
Chrome GUI Builder	3m	306	45	59.43%	97.56%
Firefox Manual	9m 15s	72	11	37.43%	76.13%
Firefox GUI Builder	5m	1232	12	30.92%	75.38%

Table 1: Rule Statistics

met.

Using non-visual rule creation method, we decided the process to create rules would be to obtain a complete list of GUI data unique to events, since this would offer complete coverage over all the data thus allowing me to retain little knowledge of sequence context or even more general data-set context. This would also allow us to focus on pairing down large data-sets into smaller sets of actual rules with no side-effects on policy coverage. Due to the small size of the data-set it was a trivial task to scroll through all the flows and copy over all unique event data and then turn them into rules. For this first test, the process of copying the GUI data off all the flows was still a manual process of click-and-drag to make selections on hidden tables of GUI values. This, however tedious it may have been, was shadowed by the amount of time it took to visually remove non-unique event data. At the end of almost 29 minutes I had arrived at a list of rules that gave me complete coverage with no contextual knowledge of how they would apply to the flows at large (or even what the data itself meant).

Using the GUI builder took 3 minutes of clicking to find the right selection of events to gain complete coverage on non-empty sequences. It involved more clicks than necessary, as I tested which events would lead towards a gain in sequence coverage and not just event coverage. After about 3 minutes of clicking I had obtained a list of rules giving complete coverage to non-empty sequences. Since this task took considerably less time than the first part of this experiment, I decided to take on some more tasks. One of those was to look up some of the GUI data values that were more interesting. One of them, "CN=Chrome_WidgetWin_1", seemed like an important value and looking it up suggested that the "_1" part of the value may be specific to the data we collect and that removing that part would make the rule more generic to future flows.

It became apparent through running this small experiment, that this process of taking the list of rules and making them a minimal set of non-test-specific rules requires a minimal understanding of how applications name and use their graphical elements. Otherwise we are forced to create longer sets of rules that give us complete coverage on the data we collect to create rules. This lead to the conclusion that work-flows (and data sets) should be small and quick so that analysts in charge of creating those rules have the time to explore the application space of those rules and have a better understanding of the decision they were making. Therefore, we decided to implement some changes to automate part one of the experiment by writing a python script to fetch a list of unique event data to get the user to the exploration part of analysis faster. Since data copied from the PEACE management application is basically a long list of GUI data with one event per line, the data cleaning and unique process is short and consists of removing empty lines, sorting the remaining lines and running 'uniq' on the sorted list of events. The remaining time, can be spent by the analyst removing collection-specific data values, such as events that include website names.

For the second experiment Firefox data was used. The process for part one used scripting instead of manual analysis to get a list of rules for manual editing. The process for part two (i.e. the GUI builder) was largely the same. This time, the case study differed in the information lost from how the data is presented for analysis. For one, since I either manually or programatically skimmed through all the flows to obtain a complete list of all unique events, I know that the final list will get me complete coverage not just to all

non-empty sequences, but to all non-empty events. However, in this case we are essentially "over-fitting" our rules to the data we collected. We are restricted in the sense that unless we change a rule to be more generic (i.e /doc/ instead of /document/) we don't know how changes we make will effect our coverage. We've lost the context of how the rules apply to sequences and flows in general in order to make the trade-off of time efficiency. In essence, the GUI builder was made to dial-back that trade-off and preserve time-efficiency as well context. This loss of context is what contributes to the only real difference in table 1, which is the number is the number of rules, and thus the percentage match of events.

When using the GUI builder, we could clearly see how each new rule effected sequence coverage versus event coverage. It became a conscious decision as to include more rules for better coverage of events even when sequence coverage was maxed out. It's difficult to know whether the rules left out would effect future coverage or coverage of flows not collected due to incomplete data-sets. This could be supplemented or even completely replaced with static analysis of applications in the future, but this work focuses on giving insights to GUI hierarchies within context of rule building. What is clear from using the GUI builder is, how rule edits effect coverage, how GUI hierarchies are laid out, and that it is easier to recognize sequences tagged with data laid out in this visual manner as opposed to trying to memorized sequences to manually create rules. Either way, the task is non-trivial is requires discretion on the part of the analyst.

6.3 Rule Overlap

Rule overlap is defined as the number of GUI-data rules that apply to each event. Table 2 shows the overlap of rules for each experiment described. Considering the two different methods of building GUI-data rules manually, the visual rule builder had less overlap in both experiments. This seems to be a direct result of the explicit decision to stop adding more rules that would provide more event coverage without adding sequence coverage. As explained above, this is not a decision we can make when building rules manually since there is no easy way to determine the amount of coverage achieved when adding, removing, and editing rules. The obvious side-effect of seemingly superfluous rules is need to manually create extra firewall policies equal to the number of rules created. Effects of extra policies on the performance of the PEACE firewall overall is not explored in this work.

7 Conclusion and Discussion

We began exploring the area of data visualization and visual data mining techniques for improving the interactions within firewalls. Using an iterative design process we created a firewall management interface and several prototypes of a GUI policy builder. By trial-and-error, we narrowed down the scope of our project to include just the tools directly supportive of the task of creating policies to target the PEACE firewall's graphical user interface data. This new data is multivariate, hierarchical and semi-structured. Network analysts who want to user this new data in their firewall policies must create string matching rules cover all of the possible sequences of graphical user interface hierarchies that can be represented within an application. In other words, network analysts are building strings patterns to match the programmatic attributes of graphical components that may trigger network requests. This task is a difficult one of building a rule-set to gain complete coverage of a collection of test data and then digging into the rule-set to generalize those rules and remove dynamic test-specific values. These troubles are motivating factors in creating a visual tool allow network analysts with no prior knowledge of the underlying data structures to create rules

matching these sequences of graphical user interface components. We borrow the idea of event-sequences from [8] to build our visual data mining system.

The lessons we learned in the process of researching ideas and even building our own graphical user interface for firewall management were used in the development of our ideas directly on the existing PEACE firewall system product. We observed that to increase the chance a visual-based system will be used in real-world products they should to be directly supportive of a network analysts tasks, or even exceed the performance of basic tools for completing their tasks. Specialized visualizations that must be learned and generic overviews of data will probably go overlooked if they do not fit within the everyday work-flows of a firewall administrator. Especially if an analyst needs to find answers from their firewalls quick, they are going to go to the tools they are most familiar and comfortable with. Even though recent studies suggest system administrators aren't the command-line fanatics they are made out to be [39], we believe we should be careful not to introduce sudden, complicated visual instruments for system configuration and end-up losing the trust of users in visualization-based tools.

In this work, we were presented with the unique opportunity to actually define the policy building work-flow for future users of the PEACE firewall system, since no usable system was in place to guide users to build GUI-data rules for their policies. The tools are in place to completely by-pass our visual system, use automation tools to create GUI-data rules, and manual inspection to remove dynamic values from rule-sets. However, from our own experience, the utility of the visual system is in the learning experience. Gaining an intuition behind the complexities of the GUI-data is not as easy when simply scrolling through large sets of strings as it is when playing around with tools and seeing the results of interactions take place.

We opted for event sequences in our work to build the data mining techniques of queries, selections, and aggregations into the system. However, there are other visualization techniques we believe are worth exploring in greater detail. A packed circles or tree representation of aggregated sequences and events has the potential to show the branching factor of all the graphical user interface components that exist in the data collected. The down-side is that it can be a complex task to properly roll up sequences into a single tree and the aggregation itself can be time consuming. After all, the purpose of Bram Cappers work is the manual exploration of aggregations that would find all those branches. We see the value in letting users, whether that be the product vendor or individual PEACE clients, explore graphical user interface data to understand it's structure. With more work, the our event-sequence system can more closely mimick that of the original event-sequences research [8], conceivably allowing users to annotate individual rules or sets of rules as those that match visual components such as a call button or a URL input text-box.

It also remains to be seen as to whether these rules can be automatically generated from data collected or from static analysis of application binaries or source code. This begs the question of getting application developers involved for the purpose of graphical user interface supported verification of network requests such as with cryptographic dynamic values. Although the ideas in the prior statements are far beyond the scope of this work and not backed up by any research, they are ideas that can invalidate the need for manual exploration of GUI-data altogether.

We leave to future research the work of exploring different visualization techniques and improving upon our event-sequence system with event aggregations and rule annotations. We leave those future workers with our lessons learned in building visual systems and call for their work to integrate with real world tasks and products.

#of Rule Matches/Event	0	1	2	3	4	Average
Chrome Manual	55	109	80	0	0	1.1025
Chrome GUI Builder	99	110	35	0	0	0.7377
Firefox Manual+Automation	692	200	149	2	63	0.6835
Firefox GUI Builder	848	148	110	0	0	0.3327

Table 2: Rule Overlap

References

[1] Ehab S Al-Shaer and Hazem H Hamed. Firewall policy advisor for anomaly discovery and rule editing. In *Integrated Network Management VIII*, pages 17–30. Springer, 2003.

[2] Joaquin Garcia Alfaro, Nora Boulahia-Cuppens, and Frédéric Cuppens. Complete analysis of configuration rules to guarantee reliable network security policies. *International Journal of Information Security*, 7(2):103–122, 2008.

[3] Yair Bartal, Alain Mayer, Kobbi Nissim, and Avishai Wool. Firmato: A novel firewall management toolkit. In *Proceedings of the 1999 IEEE Symposium on Security and Privacy (Cat. No. 99CB36344)*, pages 17–31. IEEE, 1999.

[4] Steven M Bellovin. Security problems in the tcp/ip protocol suite. *ACM SIGCOMM Computer Communication Review*, 19(2):32–48, 1989.

[5] Steven M Bellovin and William R Cheswick. Network firewalls. *IEEE communications magazine*, 32(9):50–57, 1994.

[6] Jürgen Bernard, Matthias Zeppelzauer, Michael Sedlmair, and Wolfgang Aigner. A unified process for visual-interactive labeling. *International Journal of Computer Graphics*, 2017.

[7] Bram CM Cappers and Jarke J van Wijk. Understanding the context of network traffic alerts. In *Visualization for Cyber Security (VizSec), 2016 IEEE Symposium on*, pages 1–8. IEEE, 2016.

[8] Bram CM Cappers and Jarke J van Wijk. Exploring multivariate event sequences using rules, aggregations, and selections. *IEEE transactions on visualization and computer graphics*, 24(1):532–541, 2017.

[9] D Brent Chapman. Network (in) security through ip packet filtering. In *USENIX Summer*, 1992.

[10] Bill Cheswick. The design of a secure internet gateway. In *USENIX Summer Conference Proceedings*. Citeseer, 1990.

[11] G Win eld Treese and Alec Wolman. X through the firewall, and other application relays. In *Technical Summer Conference. USENIX*, 1993.

[12] Robert Gove. V3spa: A visual analysis, exploration, and diffing tool for selinux and seandroid security policies. In *Visualization for Cyber Security (VizSec), 2016 IEEE Symposium on*, pages 1–8. IEEE, 2016.

[13] Cameron C Gray, Panagiotis D Ritsos, and Jonathan C Roberts. Contextual network navigation to provide situational awareness for network administrators. In *Visualization for Cyber Security (VizSec), 2015 IEEE Symposium on*, pages 1–8. IEEE, 2015.

[14] Ngoc Anh Huynh, Wee Keong Ng, Alex Ulmer, and Jörn Kohlhammer. Uncovering periodic network signals of cyber attacks. In *Visualization for Cyber Security (VizSec), 2016 IEEE Symposium on*, pages 1–8. IEEE, 2016.

[15] Kenneth Ingham and Stephanie Forrest. A history and survey of network firewalls. *University of New Mexico, Tech. Rep*, 2002.

[16] Alan Jeffrey and Taghrid Samak. Model checking firewall policy configurations. In *2009 IEEE International Symposium on Policies for Distributed Systems and Networks*, pages 60–67. IEEE, 2009.

[17] Seny Kamara, Sonia Fahmy, Eugene Schultz, Florian Kerschbaum, and Michael Frantzen. Analysis of vulnerabilities in internet firewalls. *Computers & Security*, 22(3):214–232, 2003.

[18] Hyungseok Kim, Sukjun Ko, Dong Seong Kim, and Huy Kang Kim. Firewall ruleset visualization analysis tool based on segmentation. In *Visualization for Cyber Security (VizSec), 2017 IEEE Symposium on*, pages 1–8. IEEE, 2017.

[19] Ui-Hyong Kim, Jung-Min Kang, Jae-Sung Lee, Hyong-Shik Kim, and Soon-Young Jung. Practical firewall policy inspection using anomaly detection and its visualization. *Multimedia tools and applications*, 71(2):627–641, 2014.

[20] Paul A Kirschner. Cognitive load theory: Implications of cognitive load theory on the design of learning. *Learning and Instruction*, 12(1):1–10, 2002.

[21] Kiran Lakkaraju, Ratna Bearavolu, Adam Slagell, William Yurcik, and Stephen North. Closing-the-loop in nvisionip: Integrating discovery and search in security visualizations. In *IEEE Workshop on Visualization for Computer Security, 2005.(VizSEC 05).*, pages 75–82. IEEE, 2005.

[22] Qi Liao, Aaron Striegel, and Nitesh Chawla. Visualizing graph dynamics and similarity for enterprise network security and management. In *Proceedings of the seventh international symposium on visualization for cyber security*, pages 34–45. ACM, 2010.

[23] Alain Mayer, Avishai Wool, and Elisha Ziskind. Fang: A firewall analysis engine. In *Proceeding 2000 IEEE Symposium on Security and Privacy. S&P 2000*, pages 177–187. IEEE, 2000.

[24] George A Miller. The magical number seven, plus or minus two: Some limits on our capacity for processing information. *Psychological review*, 63(2):81, 1956.

[25] Jeffrey Mogul. Using screend to implement ip/tcp security policies. Technical report, DIGITAL EQUIPMENT CORP PALO ALTO CA NETWORK SYSTEMS LAB, 1991.

[26] Jeffrey C Mogul. Simple and flexible datagram access controls for unix-based gateways. In *In USENIX Conference Proceedings*. Citeseer, 1989.

[27] Shaun P Morrissey and Georges Grinstein. Visualizing firewall configurations using created voids. In *2009 6th International Workshop on Visualization for Cyber Security*, pages 75–79. IEEE, 2009.

[28] Fred Paas, Alexander Renkl, and John Sweller. Cognitive load theory and instructional design: Recent developments. *Educational psychologist*, 38(1):1–4, 2003.

[29] Palo Alto Networks. Palo alto networks website.

[30] Peter Pirolli and Stuart Card. Information foraging in information access environments. In *Chi*, volume 95, pages 51–58, 1995.

[31] Marcus J Ranum. A network firewall. In *Proceedings of the World Conference on System Administration and Security, Washington, DC*, 1992.

[32] Marcus J Ranum and Frederick M Avolio. A toolkit and methods for internet firewalls. In *USENIX Summer*, pages 37–44, 1994.

[33] Jonathan C Roberts, Chris Headleand, and Panagiotis D Ritsos. Sketching designs using the five design-sheet methodology. *IEEE Transactions on Visualization & Computer Graphics*, pages 419–428, 2016.

[34] Tina Seufert, Inge Jänen, and Roland Brünken. The impact of intrinsic cognitive load on the effectiveness of graphical help for coherence formation. *Computers in Human Behavior*, 23(3):1055–1071, 2007.

[35] Hadi Shiravi, Ali Shiravi, and Ali A Ghorbani. A survey of visualization systems for network security. *IEEE Transactions on visualization and computer graphics*, 18(8):1313–1329, 2011.

[36] Richard E Smith. Sidewinder: Defense in depth using type enforcement. *International Journal of Network Management*, 5(4):219–229, 1995.

[37] Melanie Tory and Torsten Moller. Human factors in visualization research. *IEEE transactions on visualization and computer graphics*, 10(1):72–84, 2004.

[38] Tung Tran, Ehab S Al-Shaer, and Raouf Boutaba. Policyvis: Firewall security policy visualization and inspection. In *LISA*, volume 7, pages 1–16, 2007.

[39] Artem Voronkov, Leonardo A Martucci, and Stefan Lindskog. System administrators prefer command line interfaces, don't they? an exploratory study of firewall interfaces. In *Fifteenth Symposium on Usable Privacy and Security 2019)*, 2019.

[40] Simon Walton, Eamonn Maguire, and Min Chen. Multiple queries with conditional attributes (qcats) for anomaly detection and visualization. In *Proceedings of the Eleventh Workshop on Visualization for Cyber Security*, pages 17–24. ACM, 2014.

[41] Lihua Yuan, Hao Chen, Jianning Mai, Chen-Nee Chuah, Zhendong Su, and Prasant Mohapatra. Fireman: A toolkit for firewall modeling and analysis. In *2006 IEEE Symposium on Security and Privacy (S&P'06)*. IEEE, 2006.

Appendix: Sketches

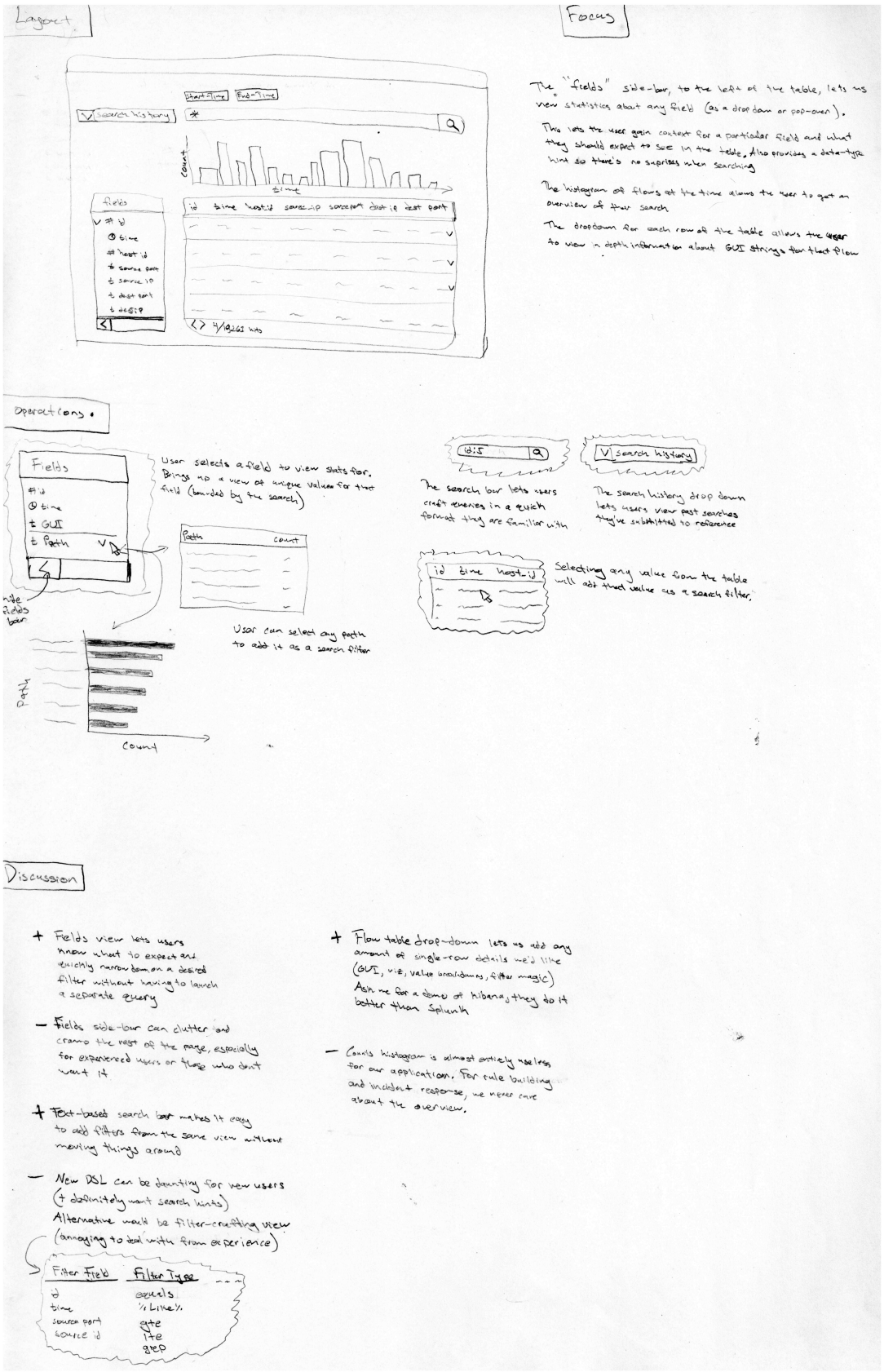


Figure 15: Design Sheet Example

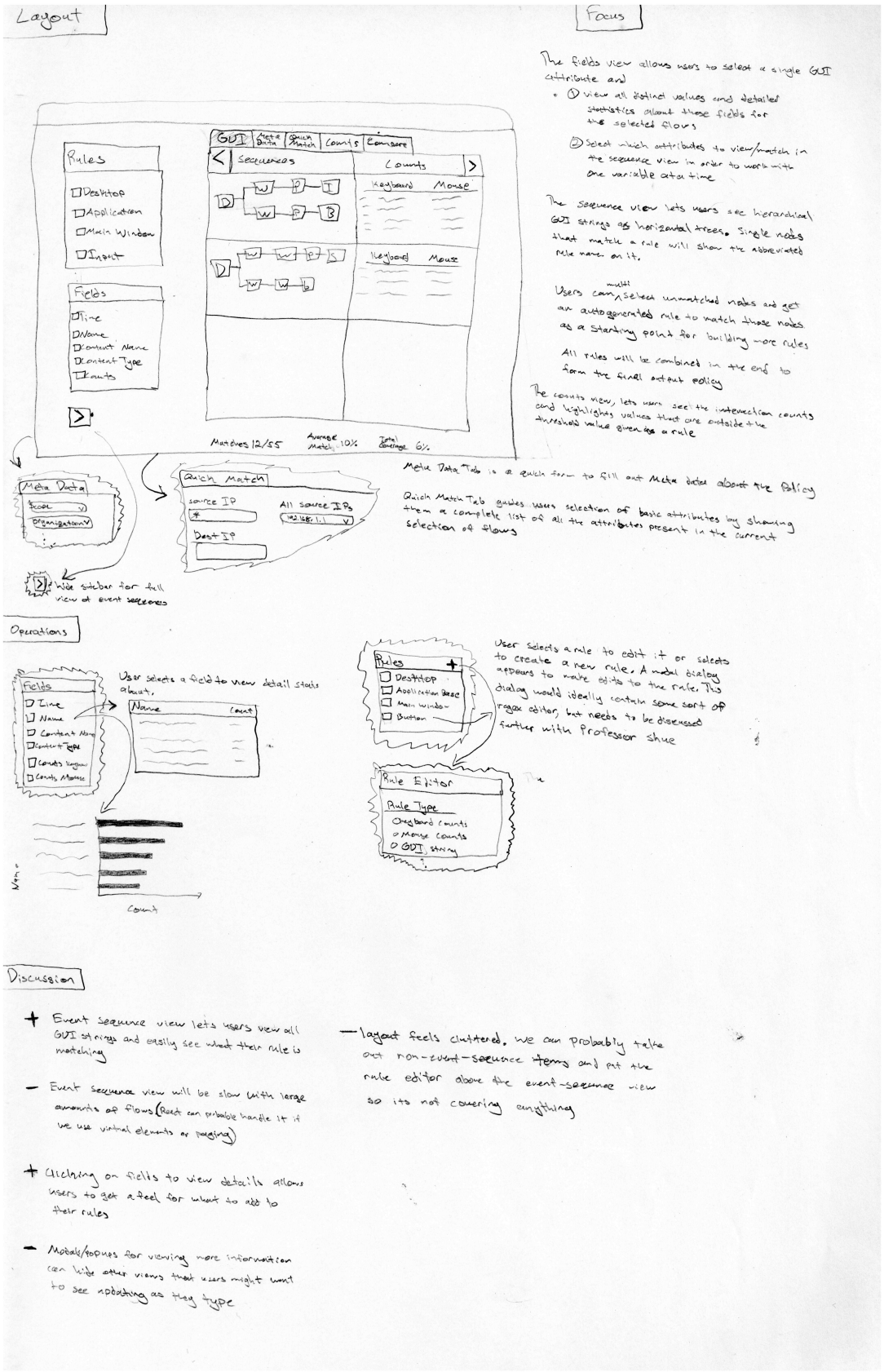


Figure 16: Design Sheet Example

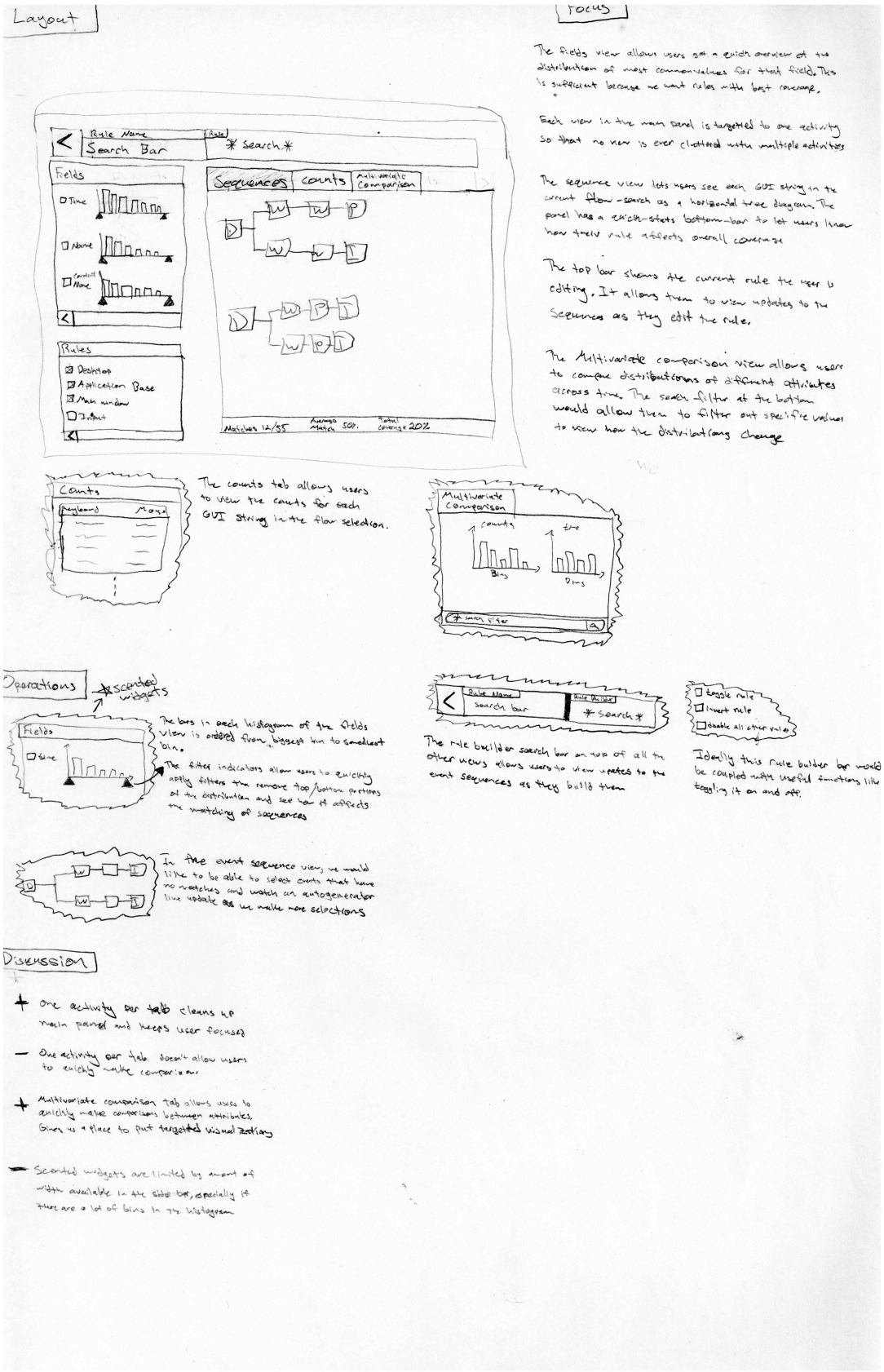


Figure 17: Design Sheet Example

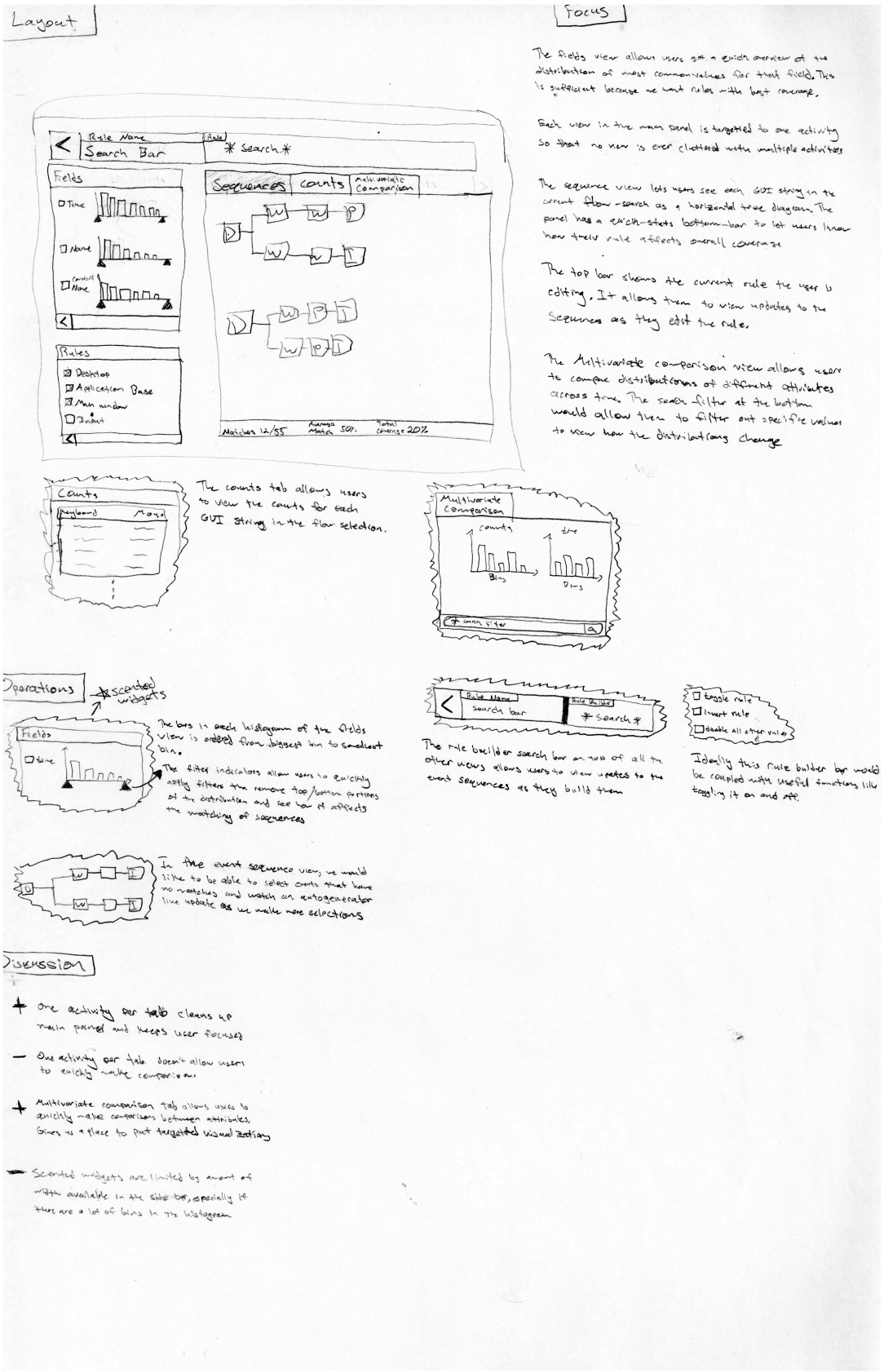


Figure 18: Design Sheet Example

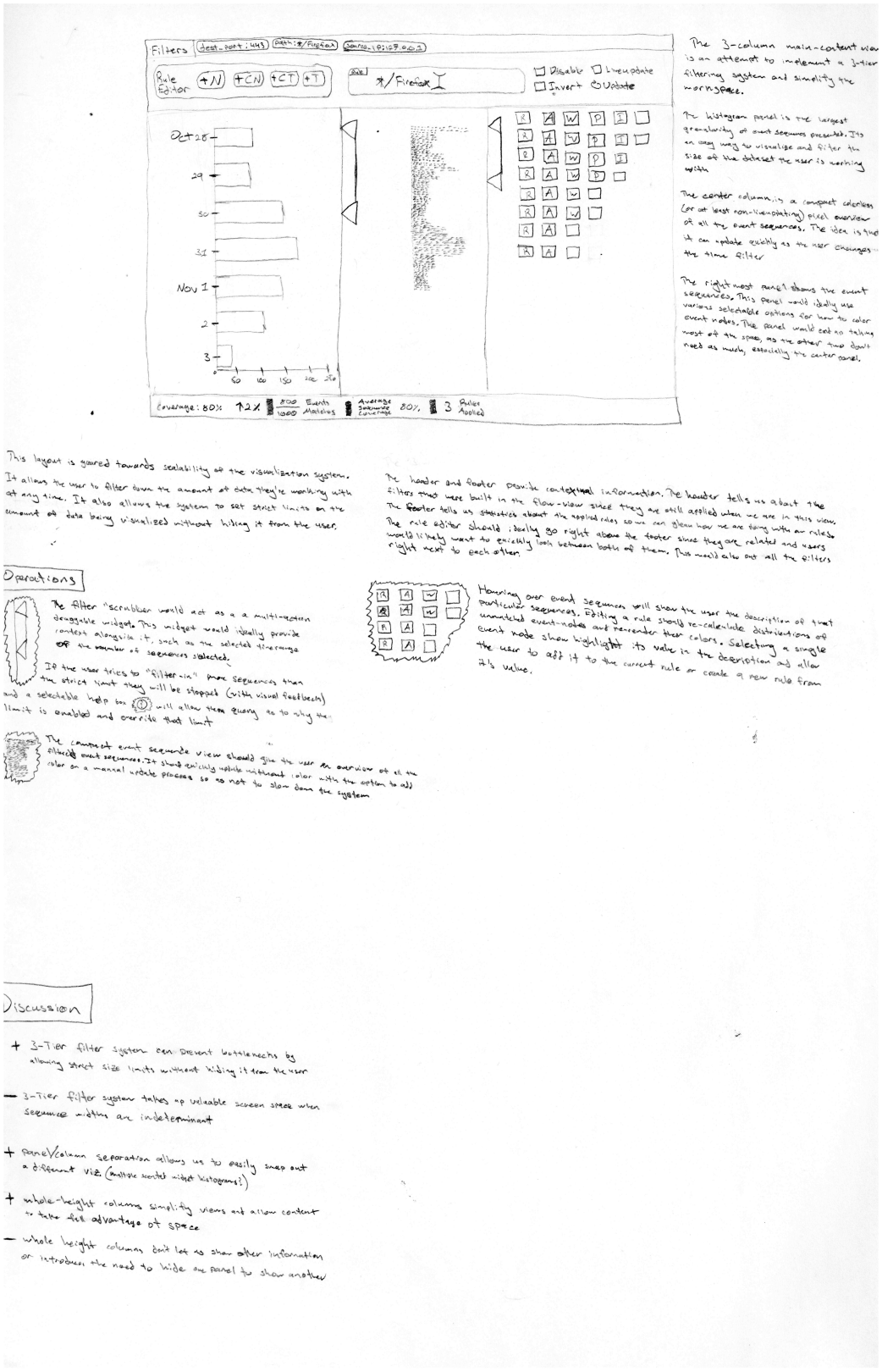


Figure 19: Design Sheet Example

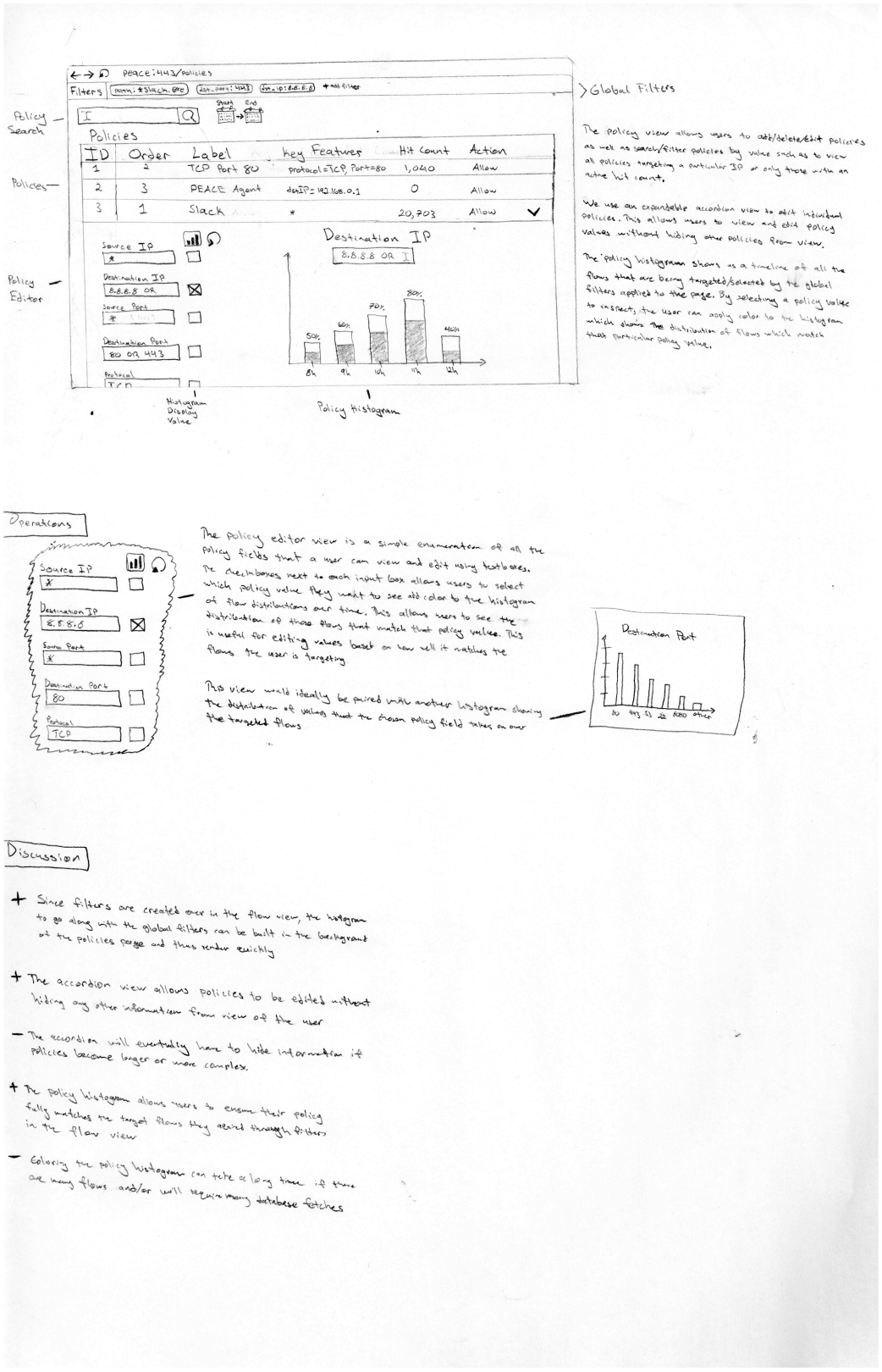


Figure 20: Design Sheet Example

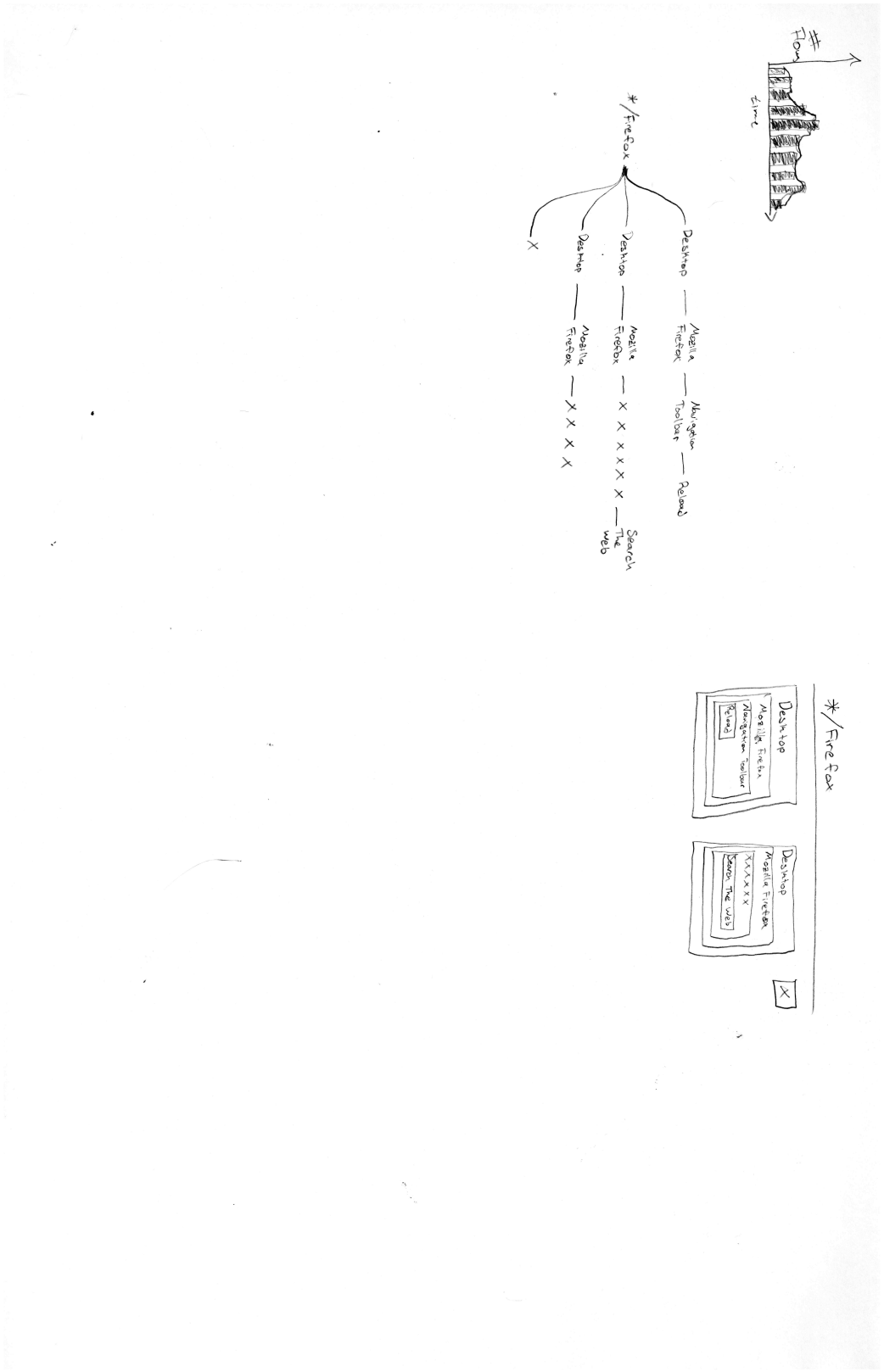


Figure 21: Notebook Sketch

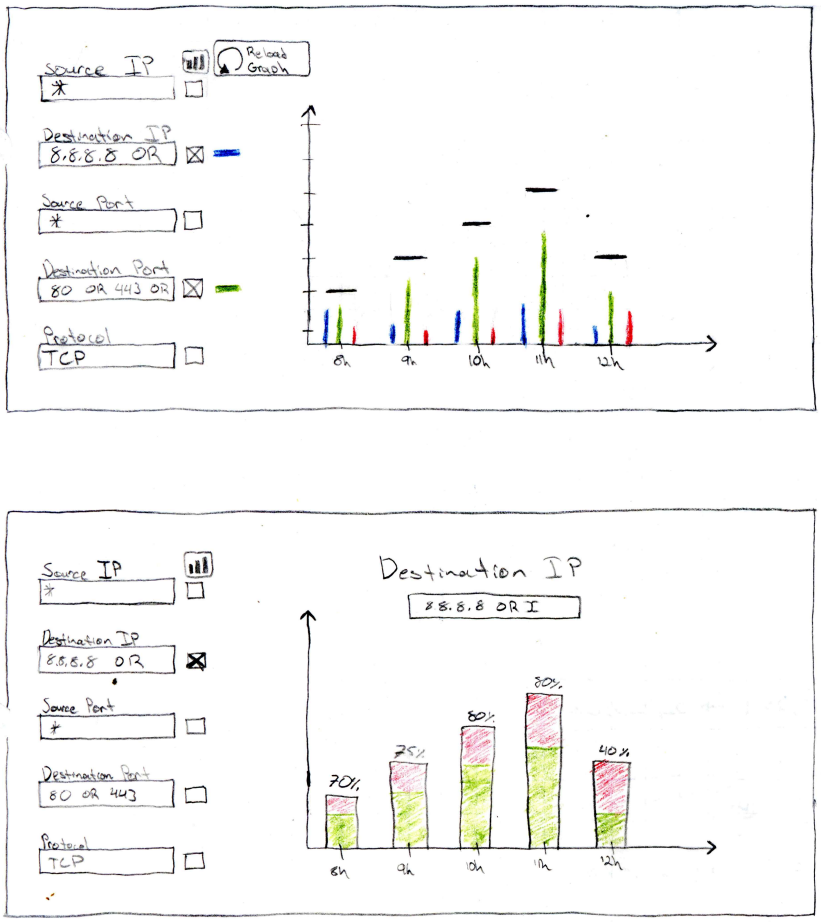


Figure 22: Notebook Sketch

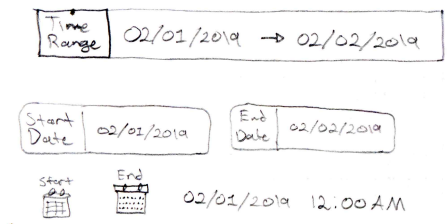


Figure 23: Notebook Sketch

Figure 24: Notebook Sketch

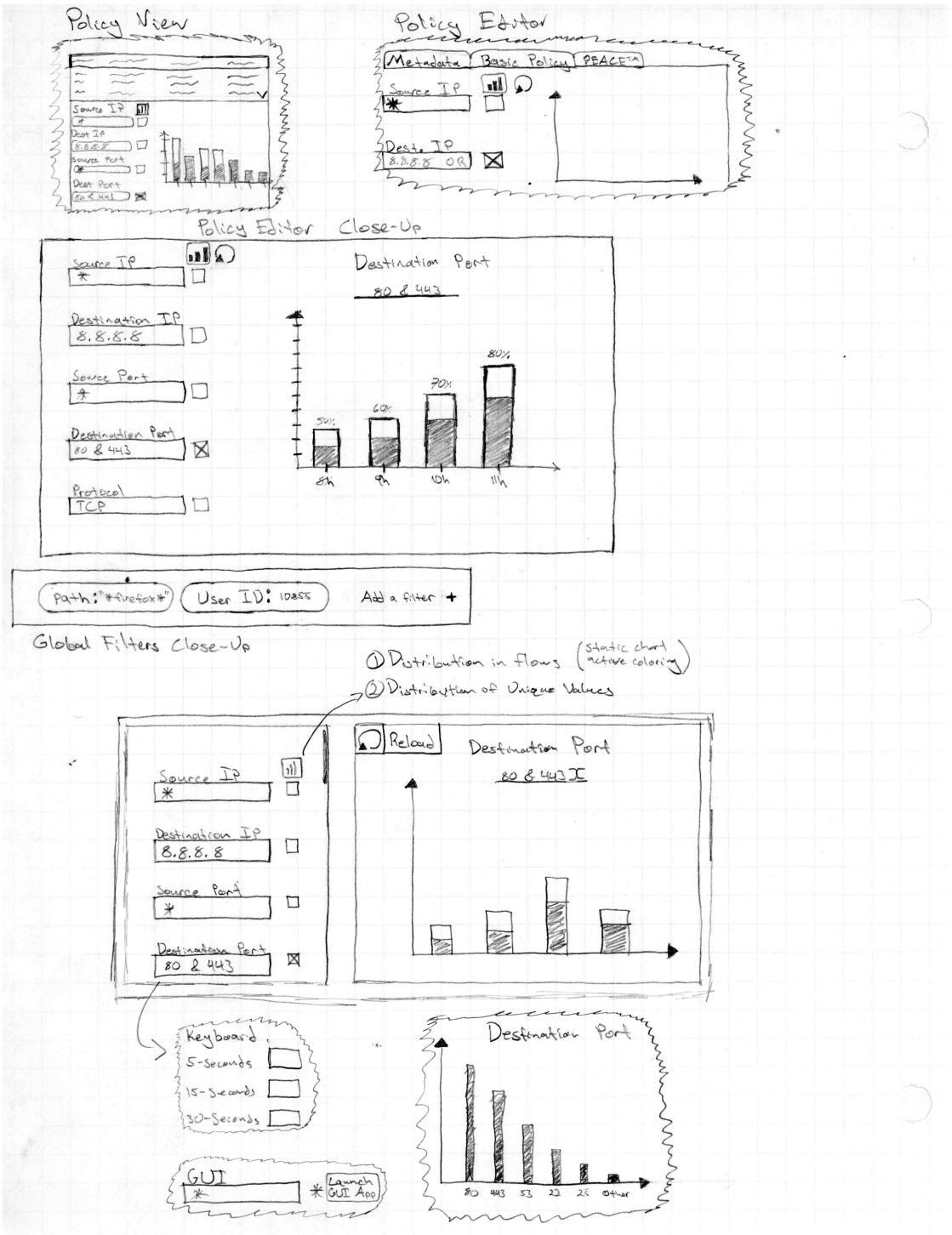


Figure 25: Notebook Sketch

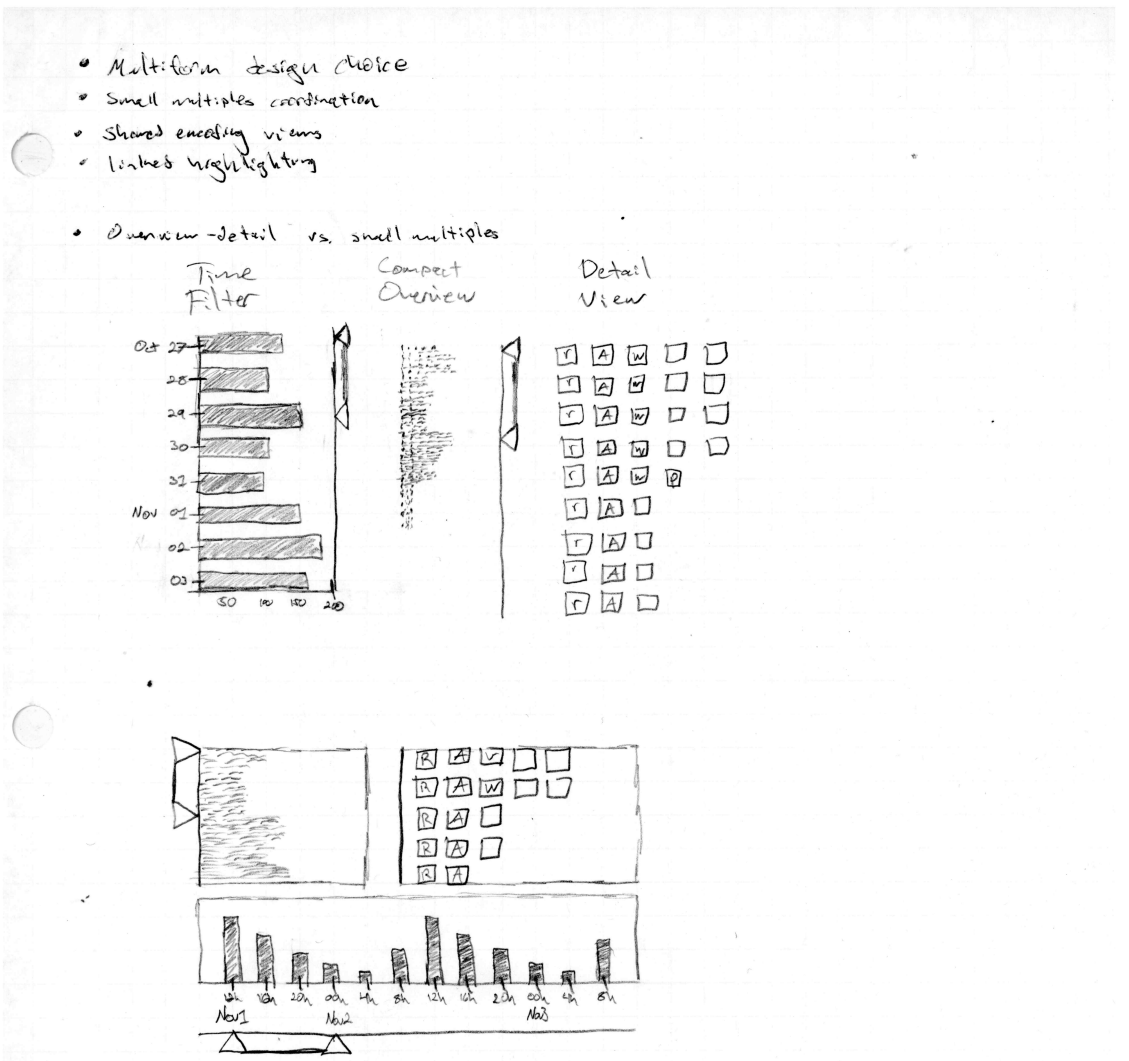


Figure 26: Notebook Sketch

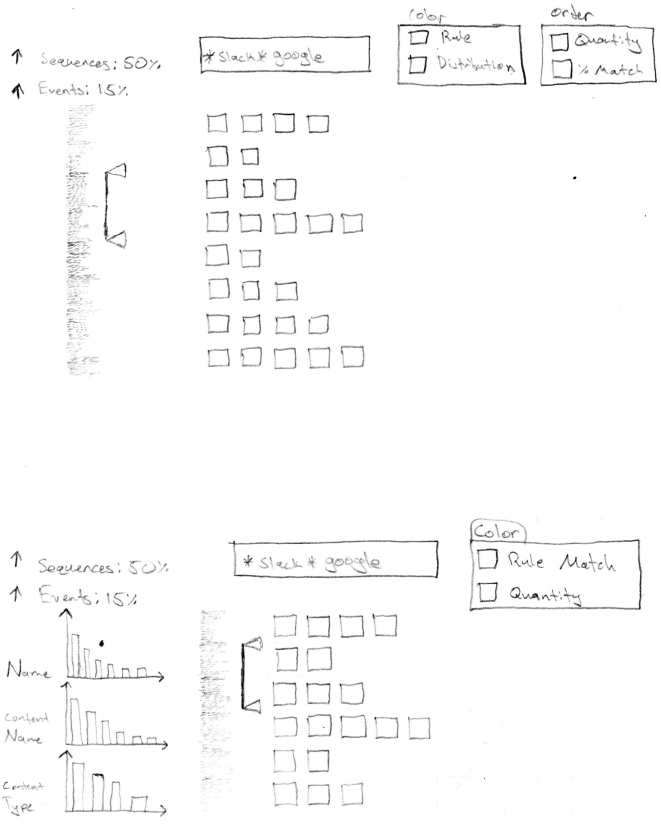
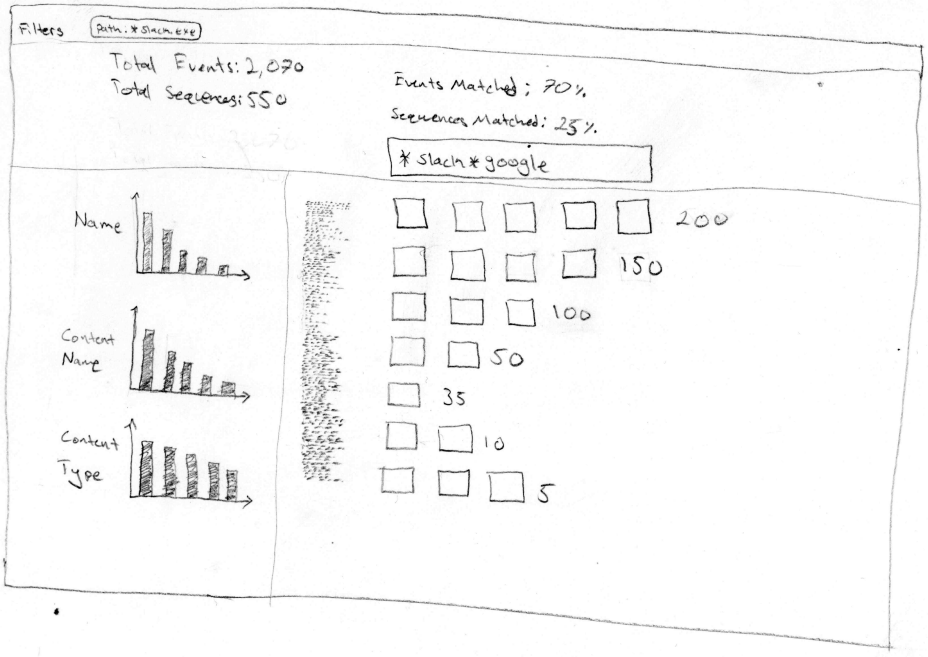


Figure 27: Notebook Sketch


Filters
path: *slack.exe
Total Events: 2,070
Total Sequences: 550
Events Matched: 70%
Sequences Matched: 25%
*slack*google
Name
Content Name
Content Type
200
150
100
50
35
10
5